8

REVISION INSTRUCTIONS AND MANUAL HISTORY

Publication No.: 210-140000-000

Publication Title: MODCOMP CLASSIC Central Processor, Reference Manual

Purpose of Revision: Reissue: Manual updated to include CLASSIC 7870 Computer.

REVISION INSTRUCTIONS: Delete and add page(s) as shown in the following table.

DELETE	ADD
N/A	N/A

NOTE: Revised pages are marked with the Revision Number in the upper unbound corner. When the manual is reprinted, this revision and all outstanding revisions will be incorporated.

MANUAL HISTORY:

REV. NO.	DATE ISSUED	REV. NO.	DATE ISSUED
Initial Issue	June 1978		
Reissue	July 1978		

(CONT'D.)

REV. NO.	DATE ISSUED	REV. NO.	DATE ISSUED

CONTENTS SUBJECT TO CHANGE WITHOUT NOTICE.

TABLE OF CONTENTS

Chapter 1. INTRODUCTION

General Description

The MODCOMP Classic 7860 and 7870 Series Computers are a high performance computer featuring a four megabyte addressing capability, single bit to sixty-four bit operand processing and a high speed fixed and floating point arithmetic unit. The 7860 Classic series will accommodate either byte parity core memory or solid state, error correcting, MOS memory with two memory protection systems. The 7870 Classic series is available with MOS memory only. An expanded set of instructions, greatly enhances the FORTRAN programming capabilities and use of the four megabyte memory capacity. All standard MODCOMP peripherals and I/O subsystems are compatible to the Classic. Selection of one or two bus I/O processors provide up to four I/O busses per system. Each I/O bus provides 16 Direct Memory Processor (DMP) channels.

Differences in Classic Models

The basic differences in Classic models are the type of memories and the I/O Processor selected.

Model	*Description*
7860 7860-E1	Classic Central Processor with: Integral Hardware Floating Point, Single Bus I/O Processor, 128K Bytes of Byte Parity Core Memory, Operators Panel, Cabinet and three spare card slots for expansion options.
7861 7861-E1	Classic Central Processor with: Integral Hardware Floating Point, Single Bus I/O Processor, 128K Bytes of Error Correction MOS Memory, Operators Panel, Cabinet and three spare card slots for expansion options.

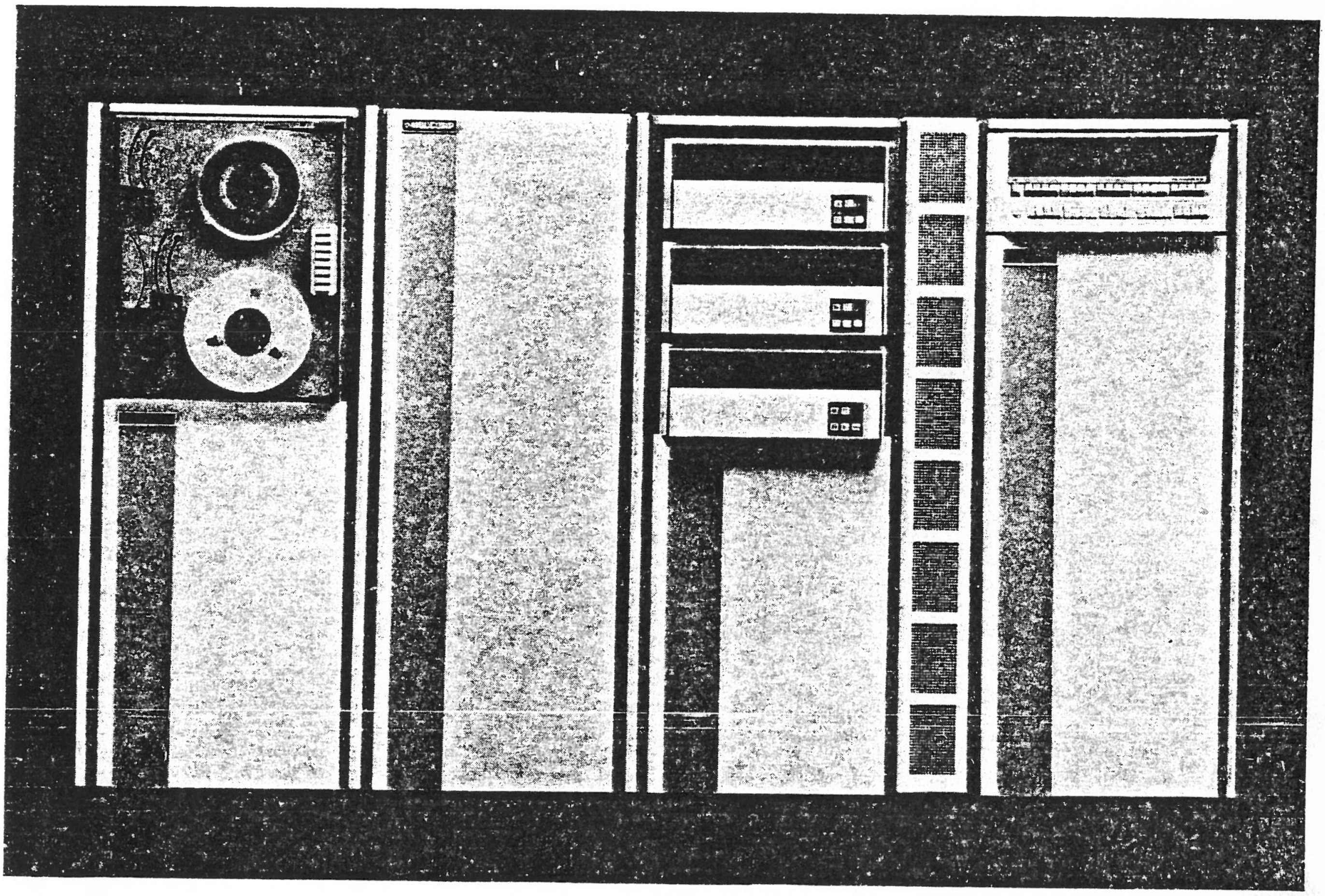


Figure 1-1. Classic System

Model	*Description*
7862 7862-E1	Classic Central Processor with: Integral Hardware Floating Point, Dual Bus I/O Processor, 128K Bytes of Byte Parity Core Memory, Operators Panel, Cabinet and three spare card slots for expansion options.
7863 7863-E1	Classic Central Processor with: Integral Hardware Floating Point, Dual Bus I/O Processor, 128K Bytes of Error Correcting MOS Memory, Operators Panel, Cabinet and three spare card slots for expansion options.
7870	Classic Central Processor with: Integral Hardware Floating Point, Dual Bus I/O Processor, 512K Bytes of Error Correcting MOS Memory, Operators Panel, Cabinet and three spare card slots for expansion options.

Chapter 2. CENTRAL PROCESSOR

Operational Modes

The Classic has three modes of operation:

- Non-Virtual
- Virtual
- Extended Addressing

Non-Virtual Mode

The operating condition of the Classic Computer after power is applied, is the Non-Virtual, Non-Pipelined mode. In this mode, all of the non-virtual operating capabilities are available including a 32-bit integer, 64-bit floating point, stack push and pull operations, condition code operations and instruction execution in registers. The standard Classic interleaved memory and overlapped instruction execution combine to provide a 125 nanosecond effective memory cycle time.

The Memory Management System and Register Context File are inactive in this mode. As a result, all memory addressing is direct (128K bytes maximum) and register context redundancy operations are inhibited.

Virtual Mode

The Virtual Mode provides full Classic capability in the form of:

- Expanded main memory for larger programs and an increased number of concurrently active tasks.
- Reduced operating system overhead and task throughput time.
- Software compatibility with existing MODCOMP II programs.
- Increased computation power and decreased instruction execution time.

Entering Virtual Mode

The Virtual Mode is established by execution of the privileged EVMO (**Enter Virtual Mode of CPU Execution**) hardware control instruction. This operation activates the Memory Management and Register Context subsystems and makes available the full operational capability of the Classic.

Memory Management System

The activated Memory Management System provides up to 4M bytes of page-structured main memory. Automatic virtual address mapping of this paged memory system using seven Task Memory Maps greatly minimizes Classic system operation overhead for task space and address management. Task virtual address translation to actual page addresses occurs at no increase in instruction execution time.

Register Context File

The activated Register Context File is comprised of 16 General Purpose Register Blocks which automatically track and restore the contents of the General Purpose Register File for the 12 highest priority system and user tasks. This operation also occurs at no increase in instruction execution time.

In this Virtual Mode of Operation, an addressing and register context switch between multiple memory resident system elements, can occur in only 4.5 microseconds. The standard MAX IV paged memory operating system is designated to take advantage of and control the extended Classic Memory Management and Register Context subsystems.

Relocatable Mode

Virtual Mode entry also invokes the Relocatable Mode of DMP Input/Output. In the Relocatable Mode, Transfer Count (TC) and Transfer Address (TA) parameters defining a DMP I/O operation are sourced from a relocatable table in the Operating System addressing space. In the Non-Virtual Mode a maximum of 16 TC/TA pairs are located in dedicated memory locations 60-6F/70-7F. Refer to Chapter 6 for a complete description of Classic DMP.

Extended Addressing Mode

The Extended Addressing Mode expands operand addressing beyond the 64K operand map, thereby enabling instructions to utilize the full 2-megaword memory size. This mode is entered by execution of the Enter Extended Memory Addressing (EXMA) instruction. The instruction must immediately preceed the indexed or short indexed instruction to be executed. This will invoke an Extended Memory Control register containing two eight-bit boundary addresses defining a permissible extended addressing space. The operand address of an instruction executed in the Extended Addressing Mode will be biased by the modulo 8K-word lower boundary specified in the EMCR. If the resultant address in extended memory exceeds the modulo 8K-word upper boundary specified in the EMCR, a system protect trap occurs.

The Extended Addressing Mode will be activated for only the one instruction immediately following the EXMA instruction. Interrupts and halts may not occur between an EXMA instruction and its subsequent memory reference instruction.

Operating States

The Classic operates either in the User State or the Privileged State. Either operating state applies to the Non-Virtual, Virtual and Extended Addressing Modes of operation described previously.

User State

In the User State, all instructions except privileged, processor, and memory control instructions may be executed by any user tasks. In the Virtual Mode, all system and user tasks are protected from each other. Furthermore, a task's own addressing space is assigned uniqe access rights (read/execute/write) on a 512-byte page basis.

Privileged State

The privileged State is standard for the Classic Operating system, and is selectable for user tasks.

In the Virtual Mode, memory page access rights protection is the same as for non-privileged (User State) tasks.

A task must be operating in the privileged state to perform the following operations:

- Control Priority Interrupts
- Perform Input/Output
- Halt the Central Processor
- Enter or Exit Virtual Mode of PCU execution
- Enter or Exit the Pipeline Mode of Instruction Fetching
- Initialize the Direct Memory Processor
- Control the Memory Management System
- Manipulate the Register Context File
- Manipulate the Extended Memory Control Register
- Manipulate the Memory Subsystem Control Registers

All of these Classic system elements and control operations are described in subsequent sections of this manual.

Data Operands

Data operands in the Classic are classified as follows:

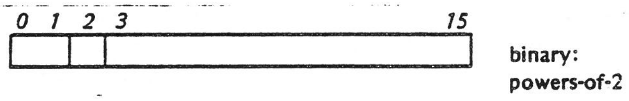
- Logical
- Character
- Virtual address
- Extended address
- Integer
- Floating point
- File
- Stack

The operands vary in length from one bit to the 256-bit stack operand and may be stored either in the general purpose register file or in memory.

Bit, byte and 16-bit operands may be specified in any register. 32-bit operands must be specified on double-register boundaries. 48-bit and 64-bit operands must be specified on quadruple-register boundaries.

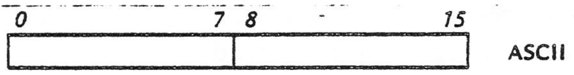
All operands in memory may be specified on any addressable (16-bit) boundary.

Logical (Bit)



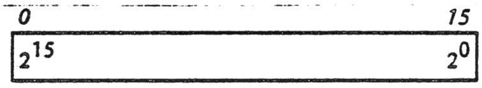
Individual bits may be addressed within 16-bit operands, both in memory and in the general purpose registers. Individual bits are numbered from 0 through 15, high-order, to low-order (left to right). Bits may be interpreted as binary values of 0 or 1 or as powers-of-two.

Character (Byte)



Individual bytes (8 bits) may be addressed within 16-bit operands, both in memory and in the general purpose registers. Individual bytes are numbered 0 and 1, or upper byte and lower byte. Character data are represented as standard 7-bit ASCII code; the high-order bit in each eight-bit byte is always a zero.

Virtual Address (16 bits)

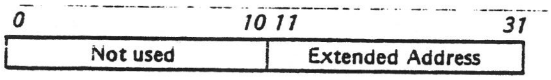


Virtual Address operands are positive 16-bit integer values without a sign. The range of values for Virtual Address words is 0 thru 65,535. This constitutes the virtual addressing range for the instructions and/or operands of individual tasks.

Virtual Effective Memory Address

Address operands are utilized in most register-memory instructions to form a Virtual Effective Memory Address (VEA). This operand type also is used in the general purpose registers by the same group of register-memory instructions to invoke "indexing" in the VEA computation. Additionally, this operation and type may be positioned in memory at the computed VEA if "indirection" is specified, in order to locate the final effective memory operand.

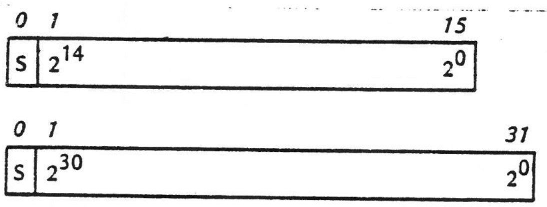
Extended Address (21 Bits)



Extended address operands are positive 21-bit integer values without a sign. The range of values for extended addressing word is 0 thru 2,097,151. This constitutes the extended addressing range for the operands of individual tasks.

Extended address operands are utilized in most register-to-memory instructions executed in extended mode to form an Extended Effective Memory Addresss (EEA). This operand type also is used in the general purpose registers by the same group of register-memory instructions to invoke "indexing" in the EEA computation. Additionally, this operation and type may be positioned in memory at the computed VEA if "indirection" is specified, in order to locate the final effective memory operand.

Integer (16-bit and 32-bit)



2's complement (binary point)

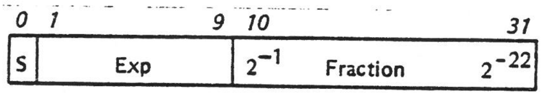
The Classic integer data format is 1 bit of sign, plus 15 or 31 bits of magnitude. For positive numbers the sign bit = 0. For negative numbers the sign bit = 1. The assumed binary point is located to the right of the least-significant bit (bit 15 or bit 31). Negative numbers are represented in 2's complement format.

32-bit integer operations are standard in the Classic. Additional 16-bit integer hardware is implemented in the Classic in order to support MODCOMP II standard 16-bit integer operations.

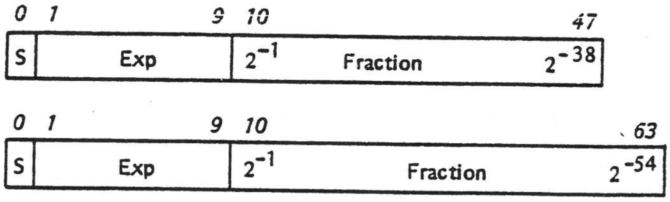
16-bit integer operations execute faster than 32-bit integer operations, resulting in a saving of time and storage for values in the range: −32,768 through +32,767.

Floating Point (32-bit, 48-bit, and 64-bit)

Three alternative formats are provided for floating point numbers in the Classic. Standard are 32-bit and 64-bit formats in which the exponent occupies 9 bits and the fraction is 22 bits and 54 bits, respectively. A compatible 48-bit format is implemented in the Classic in order to support MODCOMP II extended precision floating point operations. Here the fraction is 38 bits.



Biased binary; 2's complement



(binary point)

Each of these three formats provide an unsigned biased binary exponent with values of -256 through +255. Utilization of a binary exponent assures maximum precision for all scaling and normalization operations on the fraction.

The fraction is maintained as a signed (bit 0) binary number with an assumed binary point to the left of the most-significant magnitude bit (bit 10). Its values may be represented in the approximate range of

±1.0 x 10^7 for 32-bits, ± 1.0 x 10^{12} for 48 bits, and ±1.0 x 10^{17} for 64 bits.

Values of the exponent and the floating point number are defined as follows:

Exponent Value$_{16}$	*Floating Point Number*		*Value*
000	2^{-256}	x	Fraction Value 1>|F|>1/2
100	2^0	x	Fraction Value
1FF	2^{+255}	x	Fraction Value

The value of zero is represented by 00000...0_{16}. Hardware operations resulting in a zero fraction set the exponent to all zeros. A negative number is represented as the integer two's complement of the absolute value so that integer compare and negate operations are valid with both fixed point and floating point operands.

File (1 to 8 x 16-bits)

The general purpose registers are separable into two distinct sets or programmable files: the Lower File (R0-R7) and the Upper File (R8-R15). These independent files may be loaded and stored to or from memory in a single operation as multiples of one to eight 16-bit units, one to four 32-bit units, or one to two 48 or 64-bit units, starting with the appropriate register number and automatically including all higher numbered registers of the referenced file. For example a file reference to R4 includes R4, R5, R6 and R7.

Corresponding files in memory commence from the referenced VEA and continue for the appropriate number of additional 16-bit units, depending on the number of registers transferred.

Figure 2-1 illustrates the organization and addressing of varying length operands in general register files.

Stack (0 to 255 x 16-bits)

Last-In-First-Out (LIFO) memory stacks are implemented in the Classic. They are useful to order list-structured precedence operators and operands, and to control re-entrant and/or recursive task execution (see figure 2-2).

For each separate stack, a user task establishes a Stack Pointer Table in memory to define and maintain pointers to the low, current and high stack address, plus a stack underflow/overflow error return address.

Separate hardware PUSH and PULL instructions are provided to allocate or deallocate memory stack space and to move specified stack contents from or to the general purpose registers. These stack processing instructions address the appropriate Stack Pointer Table and supply:

- The number of words to allocate or deallocate
- The beginning register number
- The number of registers to transfer to or from the beginning of the allocated or deallocated stack space.

Register wrap-around through R0 occurs when R15 is accessed and the number of registers remaining to transfer is not zero.

Separate memory block transfer instructions are also provided to move up to 64K words between virtual and extended memory or within virtual memory.

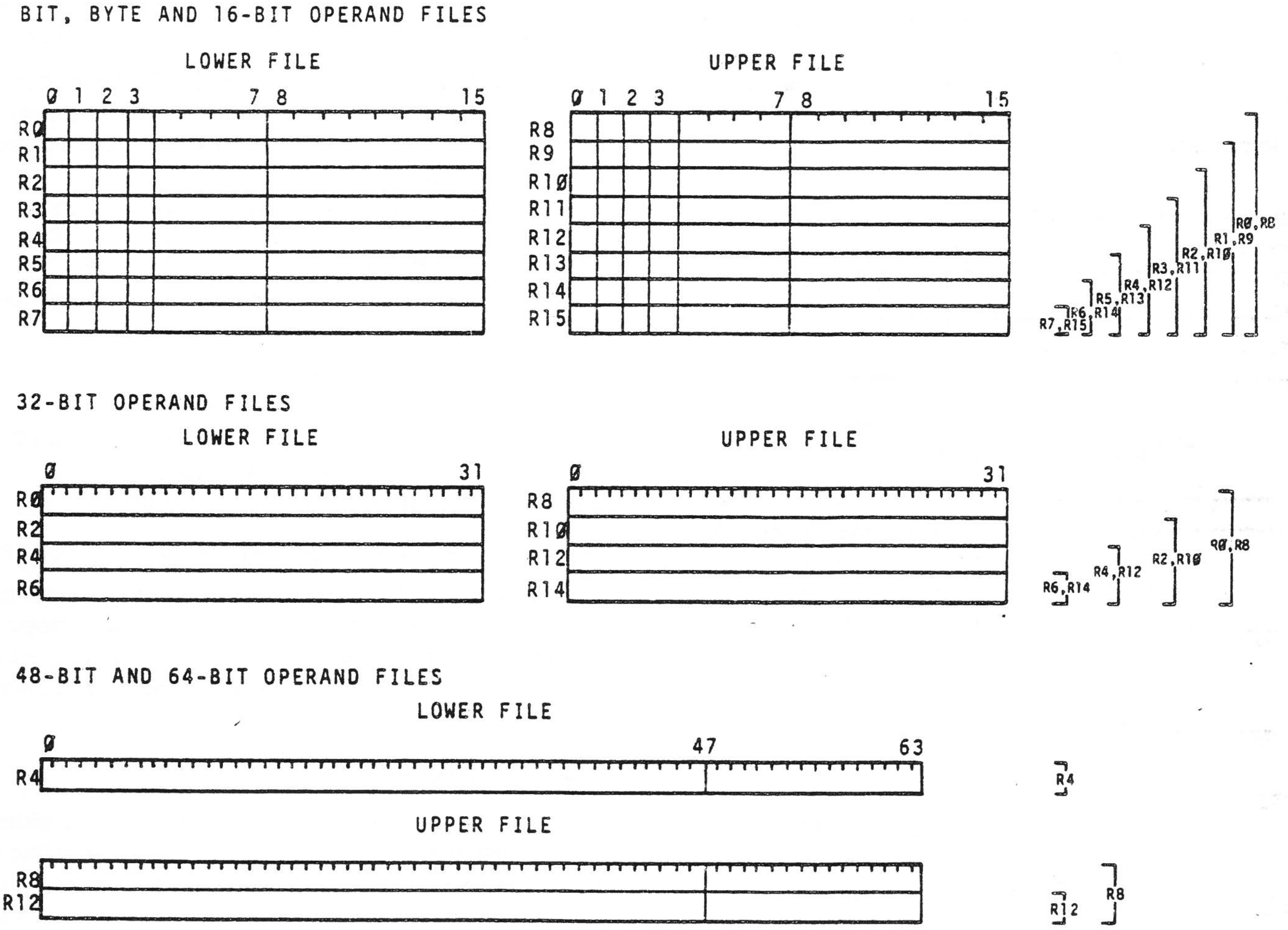


Figure 2-1. Classic Register Operand Files

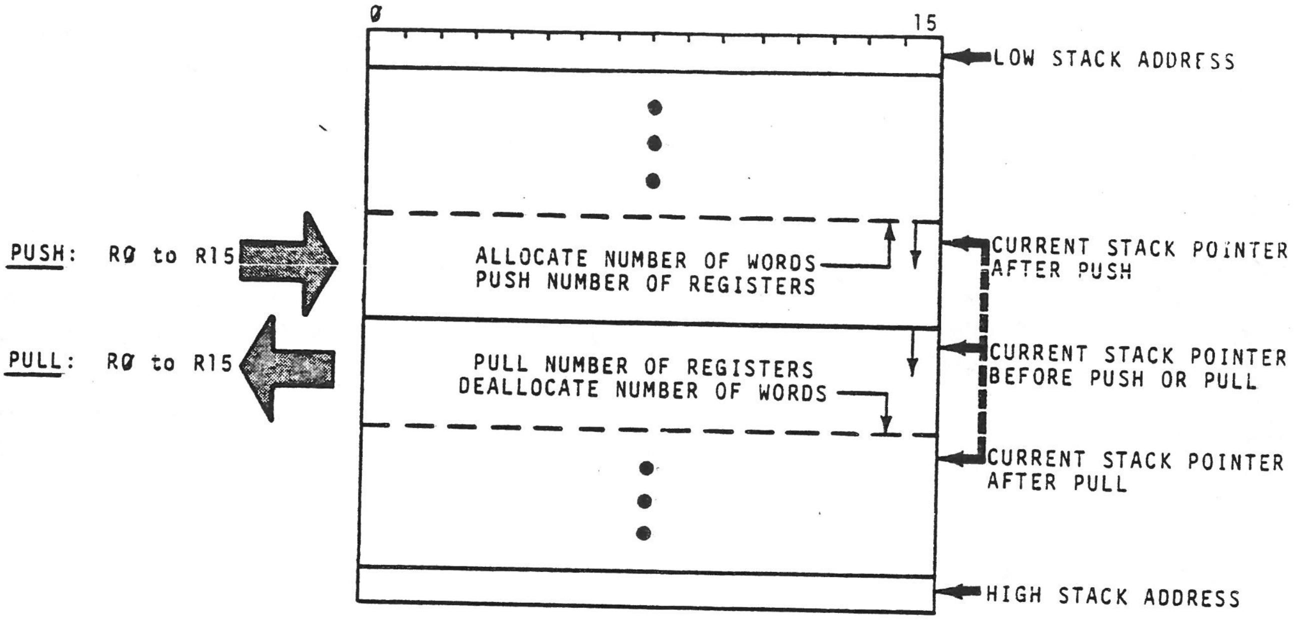


Figure 2-2. Classic Memory LIFO Stack

Program Status Doubleword (32 bits)

When priority interrupt mechanism forces a branch and save program status operation, 32 bits of status information are stored in the Program Status Doubleword. This information consists of the 16-bit Program Register for the current program, plus 16 bits of current program operational status.

The Program Status Doubleword is maintained in the following format:

Program Register (PR)	Program Status (PS)					
0 15	0 2	3	4 7	8 10	11	12 15
Program Counter (PC)	IM	PRV	GRB	OM	OH	CC
(1st Word)	(2nd Word)					

Program Register (1st Word)

PC Program Counter

Program Status (2nd Word)

IM	Instruction Map
PRV	Privilege State 0 = Unprivileged 1 = Privileged
GRB	General Purpose Register Block
OM	Operand Map
OH	Integer Overflow History
CC	Condition Codes Bit 12 – Negative Result Bit 13 – Zero Result Bit 14 – Overflow Bit 15 – Adder Carry Out

Register Addressing

Most of the Classic instructions reference one or more of the General Purpose Registers. In each case, the desired source register (Rs) and/or destination register (Rd) are designated by one or two four-bit fields (bits 8-11 and/or bits 12-15) in the format of the instruction.

Depending on the instruction operation code and other associated control bits, register addressing refers either to:

- Bit 0, 1, 2, . . . or 15 in a single register
- The upper or lower byte in a single register
- A 16, 32, 48 or 64-bit operand in from one to four registers
- A register file of one to eight registers
- A stack operand list of from one to sixteen registers.

0 7	8 11	12 15
OP Code	Rd	Rs

Where: Rd and Rs = 0-15: Any 16-bit source or destination register

Certain Classic instructions use one or two low-order bits of the four-bit register field designator to specify an operation code augment (A). This is characteristic of all 32, 48 or 64-bit operand instructions which do not address odd-number registers.

0 7	8	11	12	15
OP code	Rd	A	Rs	A

Where: Rd and Rs = 2, 4, 6 . . . 14:
Any 32-bit source or destination accumulator

0 7	8	11	12	15
OP Code	Rd	A	Rs	A

Where: Rd and Rs = 4, 8, or 12:
Any 48 or 64-bit source or destination accumulator

Many instructions use the high-order register field bit to designate indirect addressing (I) along with (or instead of) indexing which addresses only registers 1-7.

0 7	8 11	12	15
Op Code	Rs/Rd	I	Rxx

Where: Rxx = 1-7: Any 16-bit standard index register

The preceding instruction formats are shown to illustrate the standard alternative forms of addressing the general purpose registers. There are additional Classic instruction formats that reference general registers.

The description and use of all register programming instructions are described separately in Chapter 8.

Pipelined Mode of Instruction Fetching

The central processor operates in the Non-Pipelined Mode upon application of power and after any master clear operation. In this mode instructions are "pre-fetched" by a "look-ahead" mechanism (pipeline) in the central processor. Any memory write operation will cause the instruction stack (that part of the pipeline which stores the pre-fetched instructions until ready for use) to be flushed, which will result in the processor re-calling the instruction from memory. The flushing feature insures that a user task which writes into its own instruction space immediately before execution will execute properly.

The Pipelined Mode is entered by execution of a privileged EPMD instruction. In the Pipelined Mode, memory writes will *never* cause the pipeline to be flushed. This results in a significant gain in performance, but software modification of an instruction within the space of (PR)+1 and (PR)+8 will generate unpredictable results when the modified instruction is executed.

The Pipelined Mode is exited by execution of the privileged XPMD instruction which causes the instruction stack to be flushed and the next instruction to be re-fetched. Subsequent central processor operation will be in the Non-Pipelined Mode.

The EPMD and XPMD instructions are fully described in Chapter 8 of this manual.

Memory Addressing - Standard

Nine standard memory addressing modes are provided in the Classic instruction set.

Virtual Effective Address

During memory addressing, a 16-bit virtual effective address (VEA) is produced in the CPU and sent to the memory interface along with a read or write request. As specified by the operation code of the instruction being executed, either the 1, 8, 16, 21, 32, 48 or 64-bit contents of the location(s) specified by the VEA are then read from memory or replaced by the operand transferred from the Central Processor Unit.

In the Non-Virtual and Virtual Operating Modes, the VEA provides a direct virtual addressing range of 65,536 16-bit locations (or 131,072 bytes). Tasks may use separate Address Mapping Files for instructions and operands and increase the effective virtual addressing range to 262,144 bytes. In the Extended Addressing Mode, the operand addressing range becomes equal to all present memory between the lower boundary and the upper boundary of the Extended Memory Control Register (EMCR).

Memory Wrap-Around

All addressing modes in the Classic provide a memory wrap-around feature which occurs if the VEA is greater than the 65,536 word range of an active task (wrap-around to the lower memory) or, as could be the case when invoking two's complement indexing, if the resultant address is less than 0 (wrap-around to the upper memory).

Wrap-around is relative to the selected task Address Mapping File in the Virtual Mode of execution, and will always attempt to access the memory location specified by the final 16 bits of the VEA calculation, whether or not memory actually exists or is mapped at this addressed location. In the Extended Addressing Mode, a System Protect Violation interrupt is generated by an attempt to address outside of the addressing space defined by the EMCR.

Memory Addressing

The first four of the nine standard memory addressing modes are:

- Direct addressing
- Indexed Addressing
- Indirect Addressing
- Indexed/indirect addressing

They are derived from the following two 32-bit (with optional 48 or 64-bit) register-memory instruction formats:

0 7	8 11	12	15	16 31	32 47	48 63
Op Code	Rd/Rs	I	Rxx	Virtual Operand Address	Virtual Branch Address	Virtual Branch Address
					(Optional)	(Optional)

0 7	8 11	12 15	16 31
Op Code	Rd	Rxxx	Virtual Operand Address

Where:

Rd — Destination Register address
Rs — Source Register address
I — Indirect address bit, where 0 = no indirection
Rxx — Index Register address (1 ⩽ xx ⩽ 7), where 0 = no indexing Rxxx — Index Register address (1 ⩽ xxx ⩽ 15), where 0 = no indexing.

The instruction branching option utilizes bits 32-47 of the extended 48-bit format as a Virtual Branch Address. Some of the compare instructions apply these addressing modes and utilize bits 32-63 of the extended 64-bit format as two separate Virtual Branch Addresses.

Indirection applies only to the first format illustrated. These two instruction formats and their four associated memory addressing modes provide a flexible means of accessing all Classic operands, except bit and byte operands, which have their own addressing mode and are described separately.

Direct Address Mode

Direct addressing utilizes the Virtual Operand Address (VOA) as the VEA. Indexing and indirection are not involved.

If Rxx = 0 and I = 0 (first format) or if Rxxx = 0 (second format), the 16-bit address contained in bits 16-31 (VOA) of the instruction word becomes the VEA.

Equation: VEA = VOA

This addressing mode is the fundamental method in the Classic to address all 1 to 64-bit operands in memory.

Indexed Address Mode

Indexed addressing adds the contents of any one of the lower index registers (first format), or any index register (second format), to form the VEA.

If Rxx does not equal 0 and I = 0 or if Rxxx does not equal 0, the contents of the register(s) specified are added to the 16-bit address contained in bits 16-31 of the instruction word. The least significant 16 bits of the result become the VEA. The contents of the index register may be either positive or negative to produce either positive or negative displacement indexing. The indexing operation does not increase instruction execution time.

Equation: VEA = VOA + (Rxx), or VOA + (Rxxx)

This addressing mode allows lists of operands in memory to be accessed within a program "loop" by appropriately decrementing or incrementing an initialized index register (s) within the "loop".

Indirect Address Mode

Indirect addressing applies to the first format only and uses the contents of the VOA as the VEA.

If Rxx = 0 and I = 1, the 16-bit address contained in bits 16-31 of the instruction word specifies the memory location which contains the VEA. The indirect address capability is single level. One cycle time is added to instruction execution time by the indirect address word fetch.

Equation: VEA = (VOA)

This addressing mode allows operand addresses to be computed and stored in memory for subsequent use as pointers to other memory operands.

Indexed/Indirect Address Mode

Applicable to the first format only, both indexed and indirect addressing are performed.

If Rxx does not equal 0 and I = 1, the contents of register

Rxx are added to 16-bit address contained in bits 16-31 of the instruction word. The resulting address then specifies the location which contains the VEA. One cycle time is added to the instruction execution time.

Equation: VEA = (VOA + (Rxx))

The remaining standard addressing modes are:

Immediate Operand Mode

Immediate operands are expressed as 16-bit values and are located in bits 16-31 of the referencing instruction. Immediate mode instructions use the following 32-bit format:

0 7	8 11	12 15	16 31
Op Code	Rd/Rs	D	Immediate Operand

The VEA is computed, automatically, to point to the contents of the Program Register plus one, which is always bits 16-31 of the immediate instruction word.

Equation: VEA = (PR) + 1

This addressing mode is provided in a short instruction format to decrease both instruction execution time and memory storage requirements when processing logical, character, integer or address 16-bit operands that are constant for a given instruction execution.

Short-Displaced Address Mode

Short-displaced operands are bit (described separately), 16-bit, or file values located within +15 locations of the dedicated base register (R1) address. All short-displaced addressing instructions that access 16-bit values utilize the following 16-bit (or optional 32-bit, or 48-bit) formats:

0 7	8 11	12 15	16 31	32 47
Op Code	Rd/Rs	D	Virtual Branch Address	Virtual Branch Address
			(Optional)	(Optional)

Where: D = Displacement value: $0 \leqslant D \leqslant 15$

The positive displacement quantity D and the 16-bit contents of register R1 are added to generate the VEA. The contents of R1 is not modified by the VEA computation.

Equation: VEA = (R1) + D

Two instruction branching options are available which utilize bits 16-31 of the extended 32-bit instruction or bits 16-47 of the extended 48-bit instruction as the Virtual Branch Address(es).

This addressing mode is provided in a short instruction format to decrease both instruction execution time and memory storage requirements when processing operands positioned within fifteen locations of a program's base-addressed virtual operand space.

Short-Indexed Address Mode

Short-indexed operands are bit (described separately), 16-bit, 32-bit or file values located at the address specified by the contents of any one of the fifteen index registers. All short-indexed addressing instructions utilize the following 16-bit (or optional 32-bit or 48-bit) formats:

0 3 4 7 8 11 12 15 16 31 32 47

Op Code	Rd/Rs	Rx	Virtual Branch Address	Virtual Branch Address
			(Optional)	(Optional)

The contents of any one of the 15 general registers becomes the VEA. Rx specifies the register(s) which contains the VEA.

Equation: VEA = (Rx)

Two instruction branching options are available which, when selected, utilize bits 16-31 of the extended 32-bit instruction or bits 16-47 of the extended 48-bit instruction as the Virtual Branch Address(es).

This addressing mode is provided in a short instruction format to decrease both instruction execution time and memory storage requirements and to increase the number of available index registers when processing indexed operands located anywhere in a program's virtual operand space.

Byte Address Mode

Byte operands may be accessed anywhere in a program's virtual addressing space. They are addressed in an even/odd pair of selected General Registers which specify either the upper or lower byte of a 16-bit operand whose position is displaced within -16,384 to +16,383 locations of a base virtual memory address. All memory byte addressing instructions utilize the following 16-bit instruction and even/odd register pair format:

0 7	8 11	12 15
Op Code	Rd/Rs	Rx

Instruction

0 15	16	17 30	31
Virtual Base Address	S	Displacement Word	B
Even Register (Rx)		Odd Register (RxV1)	

Where:

Rx Specifies an even/odd pair of General Registers

B = 0 Specifies the Byte contained in bits 0-7

B = 1 Specifies the Byte contained in bits 8-15 of the memory location specified by the Virtual Effective Byte Address (VEBA)

The VEBA is obtained by adding the 16-bit base address to the signed byte Displacement Word (DW) which is first shifted right one bit position. This produces the VEA of the location containing the specified byte. The proper byte is then accessed from this location depending upon the state of B.

Equation: VEA = (contents of Rx) + DW
VEBA = VEA

Bit Address Mode

Bit operands may be accessed anywhere in a program's virtual addressing space, using any of the first seven standard addressing modes except immediate operand addressing. Bit operands are addressed with a bit number designator (b) that is specified in bits 8-11 of the applicable addressing format:

0 7	8 11	12 15
Op Code	b	

The addressing mode selected determines the VEA. The value of the desired bit number within the computed VEA is specified as 0 to 15, where 0 defines the high-order bit. The result is the Virtual Effective Bit Address (VEBA).

Equation: VEA = standard addressing mode
VEBA = VEA_b

Memory Addressing - Extended

An extended addressing mode, introduced in the Classic, allows access to operand bases as large as 2 megawords.

Extended addressing mode is entered by placing an EXMA instruction directly before a memory reference instruction. The EXMA will specify the high order 5 bits of the extended address.

Extended Effective Address

During extended memory addressing, a 21-bit Extended Effective Address (EEA) is produced in the CPU, according to the specified memory addressing mode, and sent to the memory interface along with a read or write request. As specified by the operation code of the instruction being executed, either the 1, 8, 16, 32, 48 or 64-bit contents of the location(s) addressed by the EEA are then read from memory or replaced by the operand transferred from the Central Processor Unit.

Memory Wrap-Around

Extended addressing provides a memory wrap-around feature which occurs if the EEA is greater than the 2,097,152 word maximum operating range (wrap-around to lower memory) or, as could be the case when invoking two's complement negative indexing, if the resultant address is less than 0 (wrap-around to upper memory).

Wrap-around is relative to the current contents of the Extended Memory Control Register (EMCR) and access will always be attempted to the memory location specified by the final 21 bits of the EEA calculation.

Extended Addressing

Extended addressing is possible when the (instruction specified) addressing mode is:

- Direct Addressing
- Indexed addressing
- Indirect addressing
- Indexed/indirect addressing
- Short-indexed addressing

The following three 32-bit or 48-bit (with optional 64- or 80-bit) register-memory instruction formats are used.

0 7	8 10	11 15	16 23	24 27	28	29 31	32 47	48 63	64 79
EXMA Op Code	0	EMA	Op Code	Rd/Rs	I	Rxx	Virtual Operand Address	Virtual Branch Address	Virtual Branch Address
								(Optional)	(Optional)

0 7	8 10	11 15	16 23	24 27	28 31	32 47
EXMA Op Code	0	EMA	Op Code	Rd	Rxxx	Virtual Operand Address

0 7	8 15	16 23	24 27	28 31	32 47	63
EXMA Op Code	0	Op Code	Rd/Rs	Rx	Virtual Operand Address	Virtual Branch Address

Where:

Rd = Destination Register Address
Rs = Source Register Address
I = Indirect bit, where 0 = no indirection
Rx = Short Indexed Register address ($1 \leqslant x \leqslant 15$), where 0 = not allowable
Rxx = Index Register address ($1 \leqslant xx \leqslant 7$) where 0 = no indexing
Rxxx = Index Register address ($1 \leqslant xxx \leqslant 15$) where 0 = no indexing

Instruction Branching

An instruction branching option is available which, when selected, utilizes bits 48-63 (first format) or bits 32-47 (third format) of the extended addressing format as a Virtual Branch Address. Some of the compare instructions apply these addressing modes and utilize bits 48-79 (first format) or 32-63 (third format) of the extended addressing format as two separate Virtual Branch Addresses.

Indirection applies only to the first format illustrated.

Extended Direct Address Mode

Extended direct addressing uses the extended memory address (EMA) and virtual operand address (VOA) as the EEA. Indexing and indirection are not involved. If Rxx=0, and I = 0 (first format), or if Rxxx = 0 (second format), then the concatenation of EMA and VOA becomes the EEA.

Equation: $\text{EEA} = \text{EMA}: \text{VOA}$

Extended Indexed Address Mode

Extended indexed addressing adds the contents of any one of the lower indexed register pairs (first format) or any index register pair (second format) to the direct address to form the EEA.

If Rxx does not = 0 and I = 0 or if Rxxx does not = 0 (Rxx or Rxxx must be an even register address), then the contents of the concatenated register pair Rxx:RxxV1 or Rxxx:RxxxV1 are added to the 21-bit address contained in bits 11-15 :32-47 of the instruction word. The least significant 21 bits of the result become the EEA. The contents of the register pair may be either positive or negative to produce either positive or negative displacement indexing.

Equation: $\text{EEA} = (\text{Rxx}):(\text{RxxV1}) + \text{EMA}:\text{VOA}$
or $\text{EEA} = (\text{Rxxx}):(\text{RxxxV1}) + \text{EMA}:\text{VOA}$

Extended Indirect Address Mode

Indirect addressing applies to the first format only and uses the contents of the VOA and VOA + 1 as the EEA.

If Rxx = 0 and I = 1, the 16-bit address contained in bits 32-47 of the instruction word specifies the memory double-word location which contains the EEA. The indirect address capability is single level. One memory double-word access time is added to the instruction execution time by the indirect address word fetch.

Equation: $\text{EEA} = (\text{VOA}):(\text{VOA} + 1)$

Extended Indexed/Indirect Address Mode

Applicable to the first format only, both indexed and indirect addressing are performed.

If Rxx does not equal 0 and I = 1, the contents of Rxx are added to the 16-bit address contained in bits 32-47 of the instruction word. The resulting address then specifies the location of the memory double-word which contains the EEA. One memory double-word access time is added to instruction execution time.

Equation: $\text{EEA} = (\text{VOA} + (\text{Rxx})): (\text{VOA} + (\text{Rxx}) + 1)$

Extended Short Indexed Address Mode

Extended short indexed operands are 1, 16, 21, 32 or 64-bit values located at the address specified by a pair of index registers. Rx specifies the register pair (Rx does not equal 0 and Rx must be an even register).

Equation: $\text{EEA} = (\text{RxV1}) : (\text{Rx})$

Two instruction branching options are available which, when selected, utilize bits 32-47 of the 48-bit instruction format or bits 32-63 of the 64-bit instruction format as the Virtual Branch Address(es).

Extended Bit Address Mode

Bit operands may be accessed anywhere in a program's extended addressing space, using any of the extended addressing modes described above. Bit operands are addressed with a four-bit, bit number designator (b) that is specified in bits 8-11 of the applicable addressing format.

Equation: EEA = extended addressing mode
$\text{EEBA} = \text{EEA}_b$

Memory Addressing, Special

Two new special memory addressing modes are provided in the Classic instruction set; memory stack addressing and relative displaced branch addressing. As in the standard memory addressing modes, a 16-bit Virtual Effective Memory Address (VEA) is produced in the Central Processing Unit as specified by the instruction operation code and the operand address elements.

Stack Address Mode

Register operands may be pushed and pulled on a last-in-first-out basis to/from dynamically allocatable memory stacks. Memory stack references utilize the four standard direct/indexed/indirect addressing modes, except the contents of the computed VEA and bits 32-47 of the special instruction format are used for stack control. All stack addressing instructions utilize the following 48-bit format:

0 7	8 11	12	15	16 31	32 35	36 39	40 47
Op Code	Rd/Rs	I		Virtual Memory Address SPT + 1		NR-1	NW

The computed VEA locates a Stack Pointer Table (SPT) in memory that contains the following five 16-bit control elements:

- Low Stack Address
- Current Stack Pointer
- High Stack Address + 1
- Overflow/Underflow Return Address
- R1 Save Area if Overflow/Underflow

The remaining stack control elements are specified in bits 36-47 of the instruction word and are defined as follows:

NW Specifies the Number of Words to allocate/deallocate to/from the stack, which modifies the Current Stack Pointer: $(0 \leqslant NW \leqslant 255)$

NR Specifies the total number of Registers to transfer: $(1 \leqslant NR \leqslant 16)$

Stack addressing is provided in the Classic as a high-speed mechanism for processing precedence-structured or hierarchial data elements, and for re-entrant or recursive task implementation.

Relative Displaced Branch Mode

Condition Codes may be tested and resultant program branching may occur to within ± 64 locations from the current value of the Program Counter. This "hop" mode of memory instruction addressing utilizes the following 16-bit instruction format:

0 7	8		15
Op Code		S	RD

Where:

RD Specifies a signed (S) Displacement value of −64 to + 63 from the current value of the Program Counter.

Relative displaced branch addressing is provided in the Classic to alter the sequence of instruction execution as a result of the occurance or non-occurance of negative, zero, overflow, or carry from the preceding instruction execution.

The short instruction format decreases both instruction execution time and memory storage requirements.

Memory Addressing - Privileged

Two privileged memory addressing modes are provided in the Classic to allow controlled access to operands located anywhere in virtual or actual memory without use of the Address Mapping Files. An appropriate 16-bit effective memory address is computed in the Central Processing Unit as specified by the instruction operation code and the operand addressing elements.

Via-Map-Image Address Mode

A memory resident "image" of an Address Mapping File may be used to access any 16-bit operand belonging to its defined virtual address space. See Chapter 8, System Programming, for detailed description of the via-map-image addressing instruction formats.

All previously described virtual addressing modes allow access only to those instructions and operands that are defined by the active Address Mapping File(s). Via-map-image addressing circumvents this restriction by allowing a privileged task to access "mapped" memory operands not belonging to the active Address Mapping File(s), or not defined in any inactive Address Mapping File (that could be made active). Any "image" in memory of an Addressing Mapping File may be utilized to access its defined operands.

Special Addressing - Privileged

Eight privileged special addressing modes are available in the Classic in addition to register and memory addressing. Five of these special addressing modes control the loading and storing of data in the Register Context File, Address Mapping Files, Map images (two modes), and Program Status Doublewords. The remaining special addressing modes control the setting and testing of the memory protect registers, interrupt latches, and input/output channels.

Refer to Chapter 8, System Programming, for specifications of these special addressing modes. These modes are available only to privileged tasks.

Instruction Set Summary

The Classic provides all of the bit, byte, file, 16-bit integer, and 32/48-bit floating point instructions of the smaller MODCOMP II, along with many useful and compatible bit, byte, and 16-bit extensions.

In addition, an entirely new set of instructions is implemented in the Classic. These new instructions provide:

- 32-bit integer and 64-bit floating point operations
- Stack push/pull operations
- Branching (and optional link) on Condition Codes
- Instruction execution in the general purpose registers
- Preparation and'control of the Address Mapping Files (and Map Images)
- Maintenance and control of the Register Context File
- Maintenance and control of the Program Status Doublewords
- Interrogation of the interrupt latches
- Direct Memory Processor initialization (for virtual address operations)
- Maintenance and Control of the Extended Memory Control Register

The complete set of Classic instructions is described in Chapter 8, System Programming. They are summarized in ten separate classifications.

- Move
- Fixed Point Arithmetic
- Floating Point Arithmetic
- Logical
- Shift
- Compare and Test
- Branch and Hop
- Control
- Interrupt and Call
- Input/Output

The first seven of these ten instruction classes use the addressing modes described previously to manipulate the large variety of bit, byte, word and multi-word operands in the general purpose registers and in memory.

Many of these instruction classes have an option to branch conditionally if the result of an operation is nonzero, if one of two arithmetic operands compares less or equal, or if the result of a test of two logical operands have one's.

Most of these instructions set Condition Codes that describe the result of an operation as being zero or negative or having produced an adder carry or overflow. The branch and hop instructions perform program branching based on any one or more of the Condition Codes being set or set by previous instruction execution. See figure 2-3 for a summary description of the functional characteristics of these computation and logic instructions.

The last three instruction classes perform unique Control, Interrupt and Input/Output functions as described in Chapter 8. Some of these control instructions also set Condition Codes.

A brief description of each instruction follows. In general, the 8-bit operation code (bits 0-7) of most of the instructions determines the operation to be performed, the operand(s) unit length with their destination and source elements, the memory addressing mode (if applicable), and the invocation of the branch option (if applicable).

Most instructions set condition codes that subsequently may be tested by the branch and hop instructions.

Move Instructions

A comprehensive set of operand transfer instructions are provided to:

- Generate bit masks in a register
- Move a byte in register(s)
- Transfer 16 to 64-bit operands between registers
- Interchange byte and 16-bit operands within the registers or between registers and memory
- Load and store bit, byte, 16, 32, 48 and 64-bit operands between registers and memory
- Push and pull stack operands between the registers and memory
- Move blocks of memory (of up to 64K words) between virtually and actually addressed memory

Fixed Point Arithmetic Instructions

Extensive fixed point arithmetic instruction capability is provided to perform add, subtract, multiply, divide two's complement and extend sign operations on 16 and 32-bit integer operands within the General Purpose Registers or between registers and memory. Bit add and subtract operations also are provided.

Floating Point Arithmetic Instructions

The high-speed floating point arithmetic instructions are hardware-implemented and provide capability to add, subtract, multiply, divide and convert to/from automatically scaled and exponentiated 32, 48 and 64-bit fraction operands, within the general purpose registers or between registers and memory.

A modified form of Memory Operand Addressing is direct with optional indexing using any of the fifteen general purpose registers.

Logical Instructions

A variety of logical instructions provide capability to extract, exclusive OR, zero and one's complement bit and 16-bit operands within the general purpose registers and between registers and memory. A new type of instruction (set register) will set or reset a 16-bit register depending on the various states of the NZOC condition codes.

Shift Instructions

The shift instructions provide capability to perform arithmetic and logical left or right shift of 16, 32 and 64-bit operands from 0 to 15-bit positions within the general purpose

Operation	Operands																									Branch							Cond Code	
	Length									Dest.			Source			Cond Code	Memory Addressing									Option					Typ			
	1 BIT	1 16 BITS	8 BITS	16 BITS	32 BITS	48 BITS	64 BITS	FILE	STACK	MEMORY	REGISTERS	CONSTANT*	MEMORY	REGISTERS	CONSTANT		STANDARD	IMMEDIATE	SHORT DISPLACED	SHORT INDEXED	STACK	DIRECT/INDEXED	TABLE INDEXED	RELATIVE DISPLACED	EXTENDED	UNCONDITIONAL	NONZERO	LESS OR EQUAL	ANY ONES	UNCONDITIONAL	CONDITIONAL	LINK RETURN	NZOC	BYTE CLASSIFY
NAME																																		
Generate		X									X				X											X							X	
Move			X								X			X											X								X	X
Transfer				X	X	X	X				X			X													X						X	
Inter Chg.			X	X							X		X	X											X								X	X
Load	X		X	X	X	X	X	X			X		X		X		X	X	X	X					X	X							X	X
Store			X	X	X	X	X	X		X				X			X	X	X	X					X									
Push									X	X				X			X				X												X	
Pull									X		X		X				X				X												X	
FIXED POINT ARITHMETIC																																		
Add		X		X	X					X	X		X	X	X		X	X	X	X					X		X						X	
Subtract				X	X						X		X	X	X		X	X	X	X					X		X						X	
Multiply				X	X		X				X		X	X			X	X	X	X					X								X	
Divide				X	X		X				X		X	X			X	X	X	X					X								X	
Two's Comp				X	X	X	X				X			X													X						X	
Ext'd Sign				X	X						X			X																			X	
FLOATING POINT ARITHMETIC																																		
Add					X	X	X				X		X	X				X					X		X								X	
Subtract					X	X	X				X		X	X				X					X		X								X	
Multiply					X	X	X				X		X	X				X					X		X								X	
Divide					X	X	X				X		X	X				X					X		X								X	
Convert					X		X				X			X											X								X	
LOGICAL																																		
Extract				X						X	X		X	X			X	X	X	X					X		X						X	
Or	X			X						X	X		X	X	X		X	X	X	X					X	X	X						X	
Exclus/Or	X			X							X		X	X	X		X	X	X	X					X		X						X	
Zero	X			X						X	X				X		X		X	X					X		X						X	
Ones Comp				X							X				X																		X	
Set Reg.				X							X					X																		
SHIFT																																		
Left Arith.				X	X		X				X			X																			X	
Rt. Arith.				X	X		X				X			X																			X	
Left Logical				X	X		X				X			X																			X	
Rt. Logical				X	X		X				X			X																			X	
Left Rotate				X							X			X																			X	
COMPARE AND TEST*																																		
Compare	X			X	X	X	X				X	X	X	X			X	X	X	X					X			X					X	
Test	X	X		X							X	X	X	X			X		X	X					X				X				X	
BRANCH AND HOP																																		
Branch										X	X						X	X		X			X							X	X	X		
Hop										X														X						X	X			

*Neither operand is modified in compare and test operations

Figure 2-3. Computational and Logic Instructions Summary

registers. A left rotate 1-bit position instruction also is provided.

Compare and Test Instructions

Instructions are provided to perform magnitude comparisons of 1, 16, 32, 48, and 64-bit arithmetic operands and to test 1 to 16-bit logical operands for any one's. Branching on result found is optional.

Branch and Hop Instructions

Unconditional and conditional branch instructions are provided, some of which save (link) the contents of the current program register prior to branching. A set of conditional hop instructions also are provided that branch relative to the current value in the Program Register.

Control Instructions

Processor Control Instructions

Processor Control instructions are provided to:

- Execute a halt
- Perform no operation
- Execute an instruction contained in a general register
- Modify the contents of the Protect Registers
- Enter and Exit the Virtual Mode of Central Processor execution
- Enter and exit the Pipelined Mode of calling instructions
- Monitor a loop execution iteration counter to control loop termination
- Enter the Extended Memory Addressing Mode

Program Control Instructions

Program control instructions are provided to

- Manipulate the contents of the Program Status Doubleword
- Load and store the Register Context File
- Manipulate the contents of Address Mapping Files and Map Images
- Load and store operands via a Map Image
- Load and store operands in Actual Pages and transfer memory (up to 64K) between extended memory and virtual memory, virtual memory and extended memory, and virtual memory and virtual memory

Interrupt and Call Instructions

Instructions are provided to:

- Interrogate and selectively enable and disable individual interrupt levels
- Activate or deactivate the priority structure at any level
- Reset the highest active level and return to the interrupted program
- Generate a priority interrupt request at any level through program control

Additional capability is provided to request an executive service and to request a multiprocessor interrupt in another CPU.

Input/Output Instructions

A standard set of I/O instructions enable byte or 16-bit transfers to be performed between the general purpose registers and up to 64 addressable peripheral or special I/O units. Command and test instructions also are provided which contain a device address and transfer a 16-bit command code or status code. A Direct Memory Processor (DMP) initialization instruction is included to assign virtual memory transfer address to DMP I/O channels.

Macro Instructions

Operation code sets are reserved in the Classic which activate the Modular Bus Control Logic. Optional logic can be incorporated to implement custom Macro instructions. These Op-codes are trapped by the Unimplemented Instruction Trap if the option is not present. The trap mechanism suspends the unimplemented instruction without attempting execution while an interrupt request is presented to the standard priority interrupt level assigned to monitor this activity.

Chapter 3. SYSTEM INTEGRITY FEATURES

Extensive hardware capabilities are provided by the Classic Series to assure the highest level of operational integrity in systems ranging in size from small dedicated processors to large multiprocessor, multiprogramming applications. These capabilities include the following:

- Power fail safe/auto restart
- Memory error checking and correction
- Memory Addressing protection
- Privileged resource protection
- Unimplemented instruction trap
- Floating Point ERROR TRAP
- Control panel disable

Power Fail-Safe/Auto Restart

The switching power supplies in the Classic processor can tolerate AC power outages up to 40 milliseconds in length without loss of DC Power. This allows the Power Fail Safe/Auto Restart Subsystem (PFS/ARS) to notify the CPU sufficiently in advance of system power failure so that system "housekeeping" may be done, allowing an orderly shut-down process.

The PFS/ARS will notify the processor and memory system of changes in power status by setting an interrupt request at level 0. In addition, the interrupt request will be forwarded to any memory system users other than the CPU so that appropriate action may be taken by them as well.

The level 0 interrupt is always enabled, so that the interrupt will normally be entered upon completion of the current instruction. During interrupt entry, the processor condition codes will be loaded by the hardware with the following information.

N: 0=EXTERNAL USER POWER STATUS CHANGE – Memory expander power status must be read to determine nature of change.
1=LOCAL POWER STATUS CHANGE

The other condition codes will be meaningful only if N indicates a local status change. If so, the following applies:

Z: 0=Power is recovering from power down
1=Power is failing

O: 0=Interrupt is to report DC power status change
1=Interrupt is to report AC power status change

Memory Error Checking and Correction

Classic memories contain error detection logic which continually monitors memory module data validity. Should an error occur, the module will trap the memory data, address and control information which generated the error, and a memory error condition will be reported both to the user for which the data was intended, and to the central processor so that appropriate corrective action may be taken.

If the memory system reports an error condition, an interrupt request at level 1 will be set. This interrupt level will be entered as soon as the priority queue allows if enabled.

If the central processor attempts to use the data which incurred the error, the current instruction will abort, and CPU activity will be suspended until the interrupt can be entered.

If an I/O Processor attempts to use the erring data, the DMP channel requesting that data will be abnormally terminated, and the device controller will (if so enabled) request a Service Interrupt.

The Classic possesses an error reporting feature which will log memory errors regardless of the memory user which incurred the error. This "global" error record is maintained by generating a level one interrupt request whenever a memory error occurs. The Classic operating software then records salient error information in response to the interrupt. Because MODCOMP IV 25/35 operating software does not include this feature, compatibility is maintained by requiring the operating system to enable interrupt level one in order to activate error reporting (a Master Clear resets the interrupt level one Enable).

It should be noted that interrupt level one disabled does not inhibit a trap to level one whenever the processor attempts to use an erring memory word. The trap will occur regardless of the state of interrupt level one Enable.

During interrupt entry to level 1, the hardware will load the condition codes with the following information.

N: 0=Global parity error has occurred, resulting in a level 1 interrupt. The stored program counter points to the next instruction to be executed when normal operation resumes and the other condition codes are meaningless.

1=The current instruction did abort. The stored program counter points to the aborted instruction, and the other condition codes contain the following information.

Z: Undefined

O,C: 0=Error was in instruction, but not in opcode word
1=Error was in instruction op code word
2=Error was in operand
3=Undefined

Should a memory error occur, the address and erring data is latched in the Plane Status Registers. A set of these registers is present on each physical memory plane and are described in detail below. Upon occurence of an error, the central processor must interrogate the memory modules via the READ MEMORY PLANE STATUS (RMPS) instructions to determine the nature of the error. Appropriate action (error logging, data correction, etc.) can then be taken and the error flag cleared.

There are two types of memory available on the Classic. Semi-conductor random access memory modules and core memory modules. Core memory modules may be programmed to detect parity errors and notify the user and CPU as described above or detection may be disabled under program control. Error detection causes no increase in the 600 core access time.

Semiconductor memory modules may be operated in either error detection mode or in error correction/detection mode. Detection mode will not attempt error correction, but will report both single and multi-bit errors. Correction/detection mode will automatically correct single bit errors without reporting them, but will continue to report double-bit errors. Both error detection and error correction may be enabled or disabled under program control. Also, the error correction mode is automatically enabled when an error is detected (detection mode enabled).

Detection mode causes no increase in the semiconductor memory access time of 500 nanoseconds and should normally be used. "Soft" errors can be trapped and corrected via software. This mode also allows the operating system to log memory errors, providing information to the service engineer.

Should a "hard" error occur, the semiconductor memory module may be placed in correction/detection mode where it will automatically correct single-bit errors without reporting them. This extends module access time to 600 nanoseconds. Multiple-bit errors will continue to be reported.

The memory module status registers may be read and written via the RMPS, RMWS, and WMS instructions. These instructions are fully described in Chapter 8, System Programming, and are outlined below to illustrate usage of the individual plane status and word status registers.

Read Memory Plane Status (RMPS)

Plane status registers, of which there are five per memory module, contain information which is applicable to the entire module, e.g., Module type, Module I.D. number and error information. Plane status registers are addressed via a register pair, selected by the Rx field of the RMPS instruction, which contains address information necessary to select a given memory module, and the status register number. These registers have the following format:

0 … 10	11 … 15	
Not Used	EMA	Rx

0 … 10	11 … 13	14 15	
Not Used	REG#	MA	RxV1

EMA the two least significant MA bits selects a memory module. The 3-bit status register number selects one of the five registers for that module.

Status register information read via the Read Memory Plane Status (RMPS) instruction will have the following format:

32 K Core Module - Plane Status Registers

0	1 2	3	4	5 7	8	9	10 12	13 15	
E	00	DM	0	000	0	PS	MOD#	100	PSR0

Where E=Error: 0 = No Error Has Occurred
1 = Error Has Occurred

DM=Detection Mode: 0 = Error Detection Disabled
1 = Error Detection Enabled

In the Error Detecting Mode, Data Validity is checked during the read data word memory cycle only.

PS=Power Status: 0 = Power Failure
1 = Power OK

MOD #= MODULE IDENTIFICATION NUMBER: Indicates the (switch selectable) identification of the memory module being addressed.

0 … 15	
Data	PSR1

If an error has occurred, this register contains the erring data word which was read from memory. If no error has occurred, the contents are undefined.

0 … 10	11 … 15	
0	EMA	PSR2

0 … 15	
MA	PSR3

If an error has occurred, PSR2 contains the extended memory address of the erring data, and PSR3 contains the memory address of the erring data. If no error has occurred, the register contents are undefined.

0	1	2 15	
UP	LP	0	PSR4

If an error has occurred, PSR4 contains the upper and lower parity bits of the erring data word. If no error has occured, the register contents are undefined.

32K or 128K Semiconductor Module-Plane Status Registers

0	1	2	3	4	5 7	8	9	10 12	13 15	
E	ME	CM	DM	BM	000	0	PS	MOD#	001	PSR0

Where:

E =ERROR : 0 = No error has occurred
1 = Error has occurred (single or multiple-bit error)

ME = MULTIPLE-BIT ERROR - Valid only if error has occurred as indicated by bit 0 = 1
0 = Non Multiple-Bit error (single-bit error)
1 = Multiple-bit error

CM= CORRECTION MODE
0 = Module will report any errors (if enabled by DM) and no attempt will be made to correct erring data
1 = Module will automatically correct single bit errors without reporting. Multiple-bit errors will continue to be reported if enabled by DM

DM = DETECTION MODE:
0 = Error Detection disabled during Read-Word operations
1 = Error Detection enabled during Read-Word operations

BM = BYTE-WRITE MODE ERROR DETECTION:
0 = Error Detection during Byte-Write operations disabled
1 = Error Detection during Byte-Write operations enabled

During a master-clear or power-up ICB, the DM and CM bits are automatically set and the BM bit is automatically reset. Error detection and correction will then be enabled only for read cycles. A WMS instruction (see below and Chapter 8) must then be executed to enable detection of erroneous memory data during a write-byte cycle (BM = 1).

PS = POWER STATUS: If power to the memory module is OK, PS = 1.
If power to the module has failed, PS = 0.

MOD #: MODULE IDENTIFICATION NUMBER: Indicates the (switch selectable) identification of the memory module being addressed.

0 15	
Data	PSR1

If an error has occured, this register contains the erring data word which was read from memory. If no error has occurred, the contents are undefined.

0 10	11 15	
0	EMA	PSR2

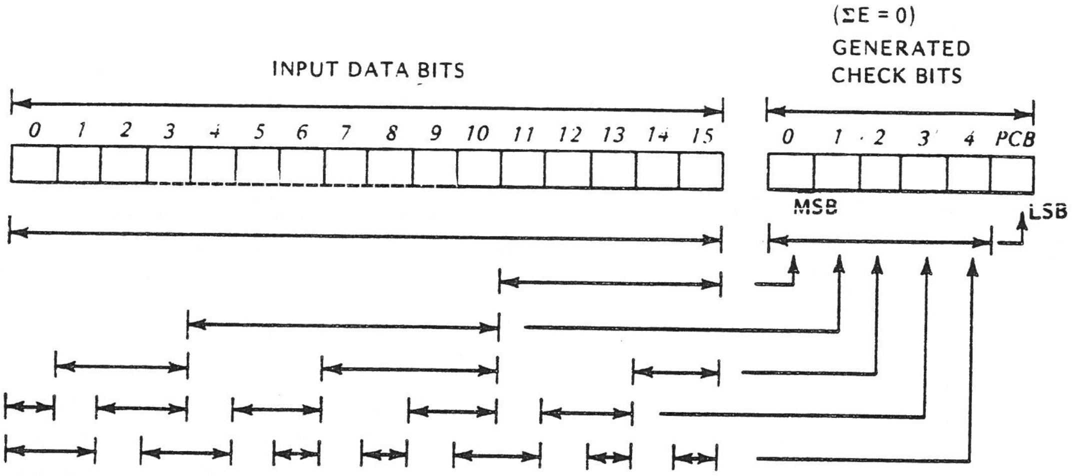
0 15	
MA	PSR3

If an error has occurred, PSR2 contains the extended memory address of the erring data and PSR3 contains the memory address of the erring data. If no error has occurred, the register contents are undefined.

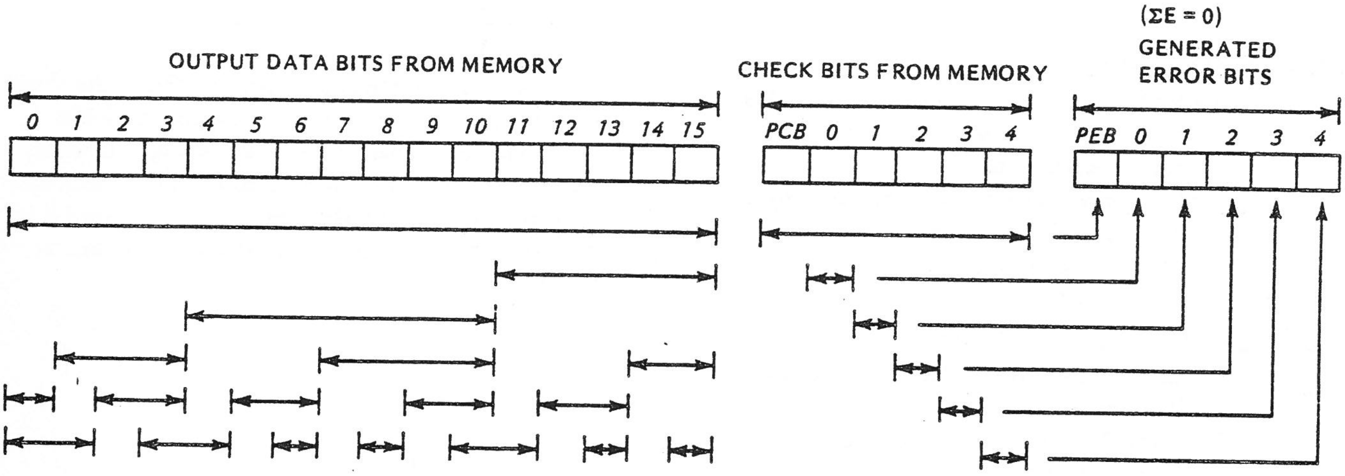
0 5	6 10	11 15	
Check Bits	Error Code	0	PSR4

If an error has occurred, PSR4 contains two fields, the check bits and error code. The check bits are the six check bits (see below) read out of memory with the erring data word. The error bits (see below) are computed from the check bits and data word, and indicate which bit of the data word is in error. If no error has occurred, the contents of the register is undefined.

The check bits are computed from the data word at write time according to the following algorithm:



The error bits are computed from the data word and check-bit field at read time according to the following algorithm:



Error bits are processed according to the following algorithm in order to determine the error condition, and in the case of a a single-bit error, generate an error code that identifies the bit location of an erred bit.

Error Bits 0-4 (Decimal)	*Error Code*		*Error Bits 0-4 (Decimal*	*Error Code*	
0	PCB	Erred	11	D6	Erred
1	C4	↑	12	D7	↑
2	C3		13	D8	
3	D0		14	D9	
4	C2		15	D10	
5	D1		16	C0	
6	D2		17	D11	
7	D3		18	D12	
8	C1		19	D13	
9	D4	↓	20	D14	↓
10	D5	Erred	21	D15	Erred

Error Bits 0-4	*PEB*	*Error Condition*	*PSR0, Bit 1 (ME-Bit)*	*Comment*
All Zero	Zero	No Error	Zero	–
All Zero	One	Parity Error	Zero	See Error Code
Non-Zero	One	Single Bit Error	Zero	See Error Code
Non-Zero	Zero	Multiple Bit Error	One	–

Read Memory Word Status (RMWS)

Word Status registers in memory (one per memory module) contain the error checking codes for data words physically addressed on that plane. They are accessed separately from the plane status registers to enable writing these bits directly into memory under program control (WMS instruction). They are addressed via the Rx field register pair, which contains the 21-bit actual address of the memory location. The address is in the following format:

0 – 10	*11 – 15*	
	EMA	Rx

0 – 15	
MA	RXV1

Data read from the memory word status registers will have the following format.

32 K Core Module - Word Status Register

0	*1*	*15*	
UP	LP	0	WSR

Where:

UP = Upper Parity Bit. This is the stored parity bit associated with the upper byte of the data word at that address. It was computed at data write time and will be set to a 1 if the data byte had an odd number of high-true (1) bits.

LP = Lower Parity bit. This is the stored parity bit associated with the lower byte of the data word at the address. It was computed at data write time and will be set to a 1 if the data byte had an odd number of high-true (1) bits.

32K or 128K Semiconductor Module - Word Status Register

0 – 5	*6 – 15*	
Check Bits	0	WSR

Where the CHECK BITS are the 6 bit error detection/correction code associated wtth the data word of that address. It was computed at data write time according to the previous illustration.

Write Memory Status (WMS)

The memory status registers (both plane status and word status) contain control and data bits which may be written under program control. They are addressed via the Rx field register pair of the WMS instruction, which contains the 21-bit actual address of the desired memory location. The address is in the following format:

0 – 10	*11 – 15*	
	EMA	Rx

0 – 15	
MA	Rxv

The data and control information to be written to the memory module should be located in the general purpose register identified by the Rs field of the WMS instruction and should have the following format:

32 K Core Module - Write Memory Status Registers

0	*1*	*2 – 9*	*10*	*11*	*12 – 13*	*14*	*15*
UP	LP	0	MDC 1	MDC 2	0	WPS	WWS

Where:

Up = Upper Parity bit. This bit will replace the upper byte parity bit associated with the upper data byte at the addressed location, if enabled by the WWS bit.

LP = Lower Parity bit. This bit will replace the lower byte parity bit associated with the lower data byte at the addressed location, if enabled by the WWS bit.

MDC 1,2 = Memory Error Detection Control:

00 = NOP
01 = Disable memory error detection
10 = Enable memory error detection
11 = NOP

WPS = WRITE PLANE STATUS:

0 = NOP
1 = Reset error flag in PSR0 of memory plane containing addressed location

WWS = WRITE WORD STATUS

0 = NOP
1 = Write UP and LP into currently addressed memory word.

32K or 128K Semiconductor Module - Write Memory Status and Register

0	6	7 8	9	10	11	12	13	14	15	
Check Bits	*	*	BDM 1	BDM 2	MDC 1	MDC 2	MCC 1	MCC 2	WPS	WWS

Where:

CHECK BITS = The contents of this field will replace the 6 bit check field associated with the data word in the addressed location, if enabled by the WWS bit.

* Spare

BDM 1,2 = Byte Detection Mode Control:

00 = NOP
01 = Disable error detection during write-byte operations
10 = Enable error detection during write-byte operations
11 = NOP

MDC 1,2 = Memory Error Detection Control:

00 = NOP
01 = Disable error detection during memory read operations
10 = Enable error detection during memory ory read operations
11 = NOP

MCC 1,2 = Memory Error Correction Control**
00 = NOP
01 = Disable error correction
10 = Enable error correction
11 = NOP

WPS = WRITE PLANE STATUS**
0 = NOP
1 = Reset Error Flags in PSR0 of memory plane containing addressed location

WWS = WRITE WORD STATUS
0 = NOP
1 = Write check bits in memory location currently addressed

**Error correction may not be disabled while the error flag is PSR0 is set. However, both error correction and the error bit may be reset by the same WMS instruction.

Memory Addressing Protection

The Classic provides extensive memory relocation and protection hardware, affording a high degree of multiprogramming capability

Entry to the virtual mode, however, enables both the virtual and extended addressing protection mechanisms.

Virtual Mode Protection

Upon entry to the virtual mode, a program's accesses to memory are made via the hardware relocation map assigned to that program. In addition, during the mapping cycle, the program's access rights to that memory location are compared to the access it is attempting. One of four access rights (AR) may be specified for the designated Actual Page:

AR Bits	*Access Rights*
00	No Access Allowed
01	Read Access Only
10	Read and Execute Access
11	Read, Execute and Write Access

If an access right violation occurs during a Read access, a violation flag bit will be attached to that word. Should the program attempt to use that word, the instruction will abort, and a system protect trap will occur.

If an access right violation occurs during a Write access, the instruction will immediately abort, and a system protect trap will occur.

In addition, when mapping via Address Mapping Files 4-7, address wrap-around will cause an access right violation. Maps 4-7 consist of 64 locations each and wrap-around occurs when bits 0 or 1 of the 16-bit VEA are equal to 1.

Extended Addressing Mode Protection

Extended addressing is defined for a program by the current contents of the extended memory control register (EMCR). This register contains two 8-bit values, the Extended Memory Lower Bound (EMCRL) and the Extended Memory Upper Bound (EMCRU). This allows extended memory to be defined with an 8K word granularity.

Extended effective addresses generated by the CPU will be added to the EMCRL to generate actual addresses. This mechanism will operate in virtual or non-virtual mode. In addition, in virtual mode, the actual address is compared to both boundary registers. Addresses either above the value of the EMCRU or below the value of the EMCRL will cause an access violation.

If the violation occurs during a Read access, a violation flag bit will be attached to the data word, and should the program attempt to use the data word, the instruction will abort and a system protect trap will occur.

If the violation occurs during a Write access, the instruction will immediately abort, and a system protect trap (level 2) will occur. If a trap occurs, the stored Program Register will point to the instruction which caused the violation.

Processor Condition Codes

During interrupt entry to the system protect routine, the processor condition codes will be loaded by the hardware with the following information.

N:	Set to 1
Z:	0 = Virtual mode addressing violation
	1 = Extended mode addressing violation
O:	0 = Instruction violation
	1 = Operand violation
C:	0 = Read violation
	1 = Write violation

Privileged Resource Protection

The Classic protects machine state registers at the instruction level. Instructions which affect certain state registers are designated as privileged instructions, and may be executed only by programs whose Program Status Word (PSW) indicates that they are privileged. This is indicated by bit 03 of the PSW being set to a 1.

In the non-virtual mode, the privileged state is always forced, hence, no privileged protection is available in that mode.

In the virtual mode, the operating system will control the PSW to allow designated programs to execute privileged instructions.

An attempt by a non-privileged program to execute a privileged instruction will result in that instruction being aborted and the generation of a system protect trap (level 2).

During interrupt entry to the system protect level, the stored Program Register will point to the aborted instruction, and the hardware will load the condition codes with the following information.

N = 0
Z = 1
O,C = Undefined

The following instructions are privileged.

Mnemonic	*Name*
CONTROL	
HLT	Halt
EVMO	Enter Virtual Mode of CPU Execution
XVMO	Exit Virtual Mode of CPU Execution
MRBM	Move Register Block Section of Context File to Memory File
MMRB	Move Memory File to Register block Section of Context File
SCRB	Select Current Register Block in PSD
LOMP	Load Operand Map Image Into Hardware Map
LIMP	Load Instruction Map Image Into Hardware Map
ZIMP	Zero Section of Instruction Map
ZOMP	Zero Section of Operand Map
SZOM	Select Map Zero as Current OM
SIOM	Select Another Program's IM as Current OM
SOOM	Select Another Program's OM as Current OM
LDVM	Load Register from Memory (Via Map Image)
STVM	Store Register Into Memory (Via Map Image)
LDAM	Load Register from (Actual) Memory
STAM	Store Register Into (Actual) Memory
RMPS	Read Memory Plane Status
RMWS	Read Memory Word Status
WMS	Write Memory Status
LXR	Load Extended Memory Control Register
RDIR	Read Internal Register
WIR	Write Internal Register
BRM	Branch to Microroutine
BRMI	Branch to Microroutine Immediate
INTERRUPT	
SIE	Set Interrupt Enable
SIR	Set Interrupt Request
SIA	Set Interrupt Active
RIE	Reset Interrupt Enable
RIR	Reset Interrupt Active
RIA	Reset Interrupt Active
CAR	Clear Active and Return
CIR	Clear Interrupt and Return
RMI	Request Multiprocessor Interrupt
INPUT/OUTPUT	
ISA,B,C,D	Input Status from I/O Group A,B,C, or D
IDA,B,C,D	Input data from I/O Group A,B,C, or D
OCA,B,C,D	Output Command to I/O Group A,B,C, or D
ODA,B,C,D	Output Data to I/O Group A,B,C or D
DMPI	Initialize Direct Memory Processor

Table 3-1. Classic Privileged Instructions

System Stall Alarm

The Classic provides a system activity timer which monitors CPU microstep executions. If a microstep fails to terminate within 50 microseconds, the instruction will be aborted and a System Protect Trap (level 2) will be generated.

During interrupt entry, the stored Program Register will point to the aborted instruction, and the hardware will load the condition codes with the following information.

N = 0 Z,O,C = undetermined

Unimplemented Instruction Trap

If an attempt is made to execute an opcode not defined for the Classic, the instruction will abort and an Unimplemented Instruction Trap (level 4) will occur.

During interrupt entry, the stored Program Register will point to the aborted instruction, and the processor condition codes will be loaded with the following information.

N:	Undefined
Z:	0 = Instruction not a REX (23XX) 1 = Instruction is a REX
O:	0 = Instruction not an EXR or EXI 1 = Instruction is an EXR or EXI
C:	Undefined

Floating Point Error Trap

Floating point operations which generate results too large or too small to be expressed in the 9-bit exponent range will abort and generate a Floating Point Error Trap (level 5). The result of the instruction will not be stored.

During interrupt entry, the stored Program Register will point to the aborted instruction, and the processor condition codes will be loaded with the following information.

N:	Set if the result was less than zero, otherwise reset
Z:	Set if the result was zero, otherwise reset
O:	Set
C:	Set if exponent underflow, reset if exponent overflow

Control Panel Disable

The control panel keylock switch may be turned to the disable position to prevent operator intervention in CPU activity. The data switches will continue to be sensed in the disabled position, but all control switches will be disabled.

Chapter 4. VIRTUAL ADDRESSING/EXTENDED ADDRESSING

Virtual Addressing

Memory addressing in the Classic is extended by the Virtual Addressing System to handle memory systems larger than 128K bytes, multiple dynamic tasks, contiguous addressing of fragmented memory, dynamic memory partitioning, and automatic address relocation.

The Classic utilizes the basic MODCOMP II compatible contiguous memory addressing mechanism until the Memory Management system virtual addressing hardware is enabled by execution of the EVMO instruction.

When enabled, the Virtual Mode of addressing does not alter or inhibit the standard addressing modes. It simply uses the final generated Effective Memory Address (EA) as a 16-bit Virtual Effective Memory Address (VEA). The VEA automatically is translated via address mapping hardware into a 21-bit Actual Page Address. This gives each program a 128K-byte virtual addressing range for its instructions and operands and allows the program to reside in any memory area.

Pipeline Operation

In both the Virtual and Non-Virtual addressing modes, the Classic CPU calls program instructions via a pipelining subsystem. This mechanism pre-calls the contents of several locations beyond that addressed by the PR (PR+1 to a maximum of PR+8), and stores them in a first-in, first-out (FIFO) stack within the CPU. This "look ahead" stack provides program access to sequential instructions without the overhead inherent in initiating a memory cycle to fetch each new instruction.

Program Branch

In cases of a program branch, CPU hardware automatically detects the branch, flushes the instruction stack and initiates a pipelined series of instruction calls beginning at the specified branch address. In the Non-Pipelined Mode (refer to Chapter 2 for a full description of Pipelined and Non-Pipelined instruction fetching) the instruction stack is also flushed and reloaded after every memory write operation. This permits memory locations that are being altered by a write operation, and that may have previously been pre-called into the instruction stack, to be recalled after the write cycle has been completed. In the Piplined Mode no memory write operations result in a stack flush, which causes a significant performance increase but will generate unpredictable results when a memory location between (PR)+1 and (PR)+8 is modified.

Extended Addressing

The extended addressing hardware further extends the Classic addressing capabilities to allow direct addressing of operand bases as large as two megawords.

The extended addressing mechanism is operational in either Virtual or Non-Virtual mode. However, no addressing protection is available when the CPU is operating in Non-Virtual mode.

When selected, the extended addressing hardware uses the 21-bit extended address generated by the central processor as an extended effective address (EEA). To this address it adds the modulo 8K-word contents of the extended memory control register lower boundary (EMCRL). This mechanism provides a bias value for extended address references which may be manipulated by the operating system, allowing multiprogramming use of the extended addressing capability.

Address Mapping and Protection

The Memory Management System utilizes two Map Select Registers, plus 8,192 page Select Registers that form seven separate Address Mapping Files (Maps).

The Map Select Registers allow either one or two Maps to be used concurrently for instructions and operands, allowing a maximum virtual addressing range of 256K bytes within a single program. The Operand Map select code in the current Program Status Doubleword (PSD) is used to select the appropriate Map for all operand and indirect address word calls. The Instruction Map select code in the PSD is used for both instruction word and address word fetches. See Section 2, Central Processor, for a description of the PSD.

The high-order 8 bits of the VEA address 1 of 256 Page Select Registers in selected Map 0, 1, 2 or 1 of 64 Page Select Registers in selected Map 3,4,5 or 6. Each Page Select Register contains a corresponding Actual Page number and Access Rights (AR) that are associated with that page.

Actual Page Assignment

The 13-bit Actual Page number points to one of a maximum of 8,192 512-byte Actual Memory Pages. Actual pages of memory need not be in consecutively numbered blocks or even in ascending order when assigned to a program's memory Map.

Page Access Rights

Access Rights are assigned to each page that is allocated to a program's virtual addressing space. Since a program's virtual addressing space is defined only by its own assigned memory Map(s), it may not access any Actual Pages not allocated to it.

Access to individual pages in a program's own virtual addressing space is classified into read, execute and/or write access. Necessarily, all of these classes of access are inhibited for all unallocated pages in a program's memory Map(s).

The 2-bit page Access Rights codes are described as follows. They are set during the page allocation function. Their monitoring and enforcement during instruction execution is described separately in Chapter 3, System Integrity Features.

AR Bits	*Access Rights*
00	No access permitted
01	Read only
10	Read and execute
11	Read, execute and write

Virtual Address Translation

The eight most significant bits (0-7) of the VEA are concatenated to either of the 3-bit Map select Register codes to address the appropriate Page Select Register within the selected Map. The addressed Page Select Register then provides the 13 most significant Actual Page Address bits (plus the two System Protect bits that define the Access Rights for the selected Actual Page (AP)). The eight least significant bits (8-15) of the VEA form the eight least significant actual Page Address bits, and are used to access the required word from the selected Actual Page.

Actual Page Allocation/Deallocation

Images of hardware Maps are constructed in memory, and then must be loaded into the hardware Maps to establish a program's virtual addressing space and page access protection information. Additional information is maintained in Map Images indicating pages that are shared between two or more programs or CPU's.

Map Images

Map Images are 512-byte memory files. They are structured similarily to the hardware Maps, contain an additional page Share code, and are located in memory on Actual Page boundaries. See figure 4-1.

During the page allocation function, Map Images are constructed in sections from the available page information. They are assigned required Actual Pages as available, are given unrestricted access and private status, and must be copied into the program's hardware Map.

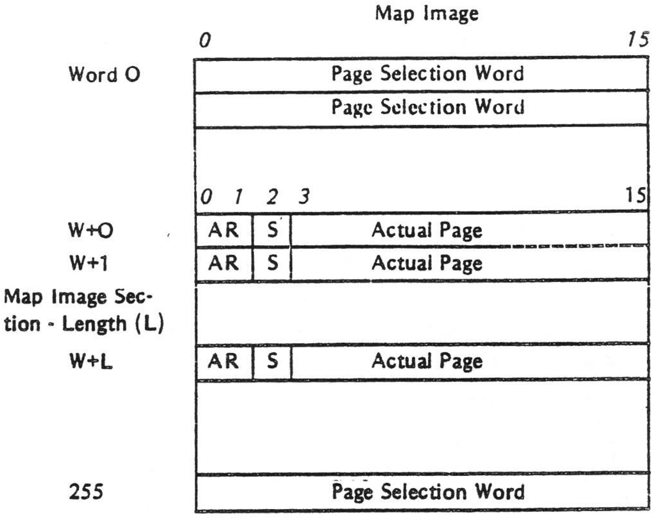


Figure 4-1. Map Image in Memory

During page deallocation, Map Image sections are processed according to their previous page allocations. Assigned actual Pages are returned to the Available Memory Pool except for any nonexistent or shared pages.

Share Code

The Share code is used in the Map Image to prevent Actual Page deallocations for pages that are shared by two or more programs and for multiprocessor shared memory configurations. Note that the Share code is not implemented in the hardware Maps.

Share Bit	*Share Code*
0	Private memory page
1	Shared memory page

Extended Address Relocation and Protection

The extended addressing mechanism utilizes a 16-bit Extended Memory Control Register (EMCR) to relocate extended addressing references and provide extended addressing memory access protection. The lower byte of the EMCR is used as a modulo 8K-word relocation bias value, and is added to the CPU generated 21-bit extended effective address.

The resulting address is compared to both the modulo 8K-word EMCR upper boundary (upper byte of the EMCR) and to the lower boundary. Attempted memory accesses above or below these boundaries will result in the instruction's being aborted and a system protect trap being generated, if the CPU is in virtual mode. No access protection is provided in non-virtual mode.

The EMCR may be loaded by execution of the LXR instruction. Access to extended memory may be inhibited by setting the Lower Boundary of the EMCR to a value greater than the Upper Boundary.

Chapter 5. PRIORITY INTERRUPTS

The Classic Priority Interrupt System contains 16 levels including the Power Fail Safe/Auto Start. Each external level can be selectively enabled and disabled under program control. Internal levels 0, 2, 4, and 5, are always enabled. In addition, the recognition of interrupt signals can be deferred for all interrupt levels below a selected level. Furthermore, interrupt request signals can be generated by instruction execution.

The I/O interrupt levels C_{16}, D_{16} have party line interrupt structure with seventeen subpriorities each and automatic source identification for up to 64 controllers.

Each priority level is assigned four dedicated memory locations for the entry and return addresses unique to that level. The entry address of the interrupt processing routine and the new Program Status Doubleword (PSD) are stored in two dedicated locations. The return address, which is the contents of the Program Register (PR), and the Program Status is stored in the other dedicated locations at the time the interrupt routine was entered. The 64 sub-levels of each I/O interrupt level share the return address and one entry address (common status word) of that level but are assigned unique entry address locations.

Nested interrupt routine execution is automatically handled for the 16 priority levels. The sub-levels of each I/O priority interrupt level cannot interrupt each other, but if several attempt to interrupt at the same time, the highest priority sub-level is recognized first.

Functional Description

The Classic Priority Interrupt Subsystem provides the mechanism to automatically alter the programmed sequence of the Central Processor in response to an external stimulus (interrupt request). The alteration consists of storing the present contents of the Program Register and the Program Status in dedicated addresses unique to the priority of the external stimulus, and forcing an indirect branch through a third unique dedicated location. The priority of the interrupt stimulus is stored in the interrupt logic and is used to inhibit the servicing of lower and equal priority requests. Higher priority requests may, however, still be recognized and processed. The program is returned to the last point of interruption by resetting the current priority status, and then executing an indirect branch through the dedicated memory location associated with the current priority in which the PR was stored, and restoring the status also saved at interrupt entry.

The Priority Interrupt Subsystem is modular in implementation. It interfaces to the CPU Bus Structure as four source and three destination registers. Its interface to the control memory is synchronous in operation.

The Priority Interrupt structure will accommodate up to 16 distinct levels of priority.

In addition, the common data and service interrupt levels, utilized by the standard I/O Bus Structure, may each link to 64 different service routines, providing 142 unique interrupt entry points for external stimuli.

Interrupt Structure

The interrupt structure contains three base registers (1 each for Enable, Request, and Active), a Priority Queue, and a Dedicated Address Generator. The Enable, Request, and Active registers may be accessed by the bus structure through program control. In addition, the Request registers may be set, on a bit-by-bit basis, by external interrupt requests. The Priority Queue constantly monitors the states of the three base registers, and presents an interrupt request to the CPU interface control when the required conditions are met. In addition, the Priority Queue presents the current priority data to the Address Generator, whose function is to generate dedicated address information for the memory accesses required to enter or exit an interrupt routine.

Level Assignments

The dedicated memory locations for each interrupt level and the signals connected to these levels are shown in Table 5-1.

The lower numbered of the two locations dedicated to the return linkage is always used to store the Program Register while the higher numbered location is used to hold the Program Status as reflected at interrupt entry. Both of these locations are restored when the interrupt level exits. The lower location is transferred to the Program Register and the higher numbered location to the Classic status control as defined by the Program Status Doubleword (PSD) in Chapter 2.

Memory Location $_{16}$	*Level* $_{16}$	*Program Linkage*	*Interrupt Signal*
20,41 EN	0	Return	Power Fail Safe/Auto Start
21,40		Entry	
22,43	1	Return	Memory Parity
23,42		Entry	
24,45 EN	2	Return	System Protect
25,44		Entry	
26,47	3	Return	Multiprocessor Communications
27,46		Entry	
28,49 EN	4	Return	Unimplemented Instruction Trap
29,48		Entry	
2A,4B EN	5	Return	Floating Point Overflow
2B,4A		Entry	
2C,4D	6	Return	Real Time Clock
2D,4C		Entry	
2E,4F	7	Return	External
2F,4E		Entry	
30,51	8	Return	External
31,50		Entry	
32,53	9	Return	External
33,52		Entry	
34,55	A	Return	External
35,54		Entry	
36,57	B	Return	External
37,56		Entry	
38,59	C	Return	I/O Data Party Line
58		Entry Status	
3A,5B	D	Return	I/O Service Party Line
5A		Entry Status	
3C,5D	E	Return	Console Interrupt
3D,5C		Entry	
3E,5F	F	Return	Task Scheduler
3F,5E		Entry	

Table 5-1. Interrupt Level Assignments

Chapter 6. INPUT/OUTPUT

The Classic Models 3770 and 3771 I/O Processors perform I/O operations between up to 63 uniquely addressable peripheral controllers and the General Purpose Registers. Register I/O operations interface to the Modular Bus and are controlled by I/O instruction execution in the CPU.

In the Non-Virtual CPU mode 16 unique direct memory access channels are available; in the virtual mode 64 channels are available. Device address 00 is dedicated to the I/O processor status and may not be assigned to any controller. Direct Memory I/O operations interface to the Memory Interface Bus and are controlled by the IOP Direct Memory Processor (DMP).

The Model 3770 IOP handles a single I/O Bus and the Model 3771 handles two separate I/O buses. Each bus may be of switch-selectable length and rate. The 100 foot bus length will support an 800ns interrupt update signal and 600ns input sync; the 25 foot length will support a 400ns interrupt update and 400ns input sync. Each I/O bus will accommodate a maximum of 8 Class II DMP controllers - or 16 Class III DMP controllers.

Input/Output Interrupts

Classic Input/Output operations are interfaced to the Priority Interrupt System (See Chapter 5).

Two standard I/O Priority Interrupts are provided to initiate transfers between peripheral devices and the CPU. (See Table 6-1, Peripheral Device Interrupt Assignments.) The higher priority Data Interrupt, Level C_{16}, is used to initiate data transfers. The Service Interrupt, level D_{16}, is used to initiate service routines for end of record, error, and similar conditions.

I/O	*Interrupt Locations*		*DMP Locations*		*Device*	
Priority	*Data*	*Service*	*TC*	*TA*	*Address*	*Peripheral Device*
1	81	C1	61	71	01	Moving Head Disc
0	82	C2	62	72	02	Fixed Head Disc
2						High Level Analog Input Subsystem
	90		60	70	10	– Channel Output
	91	D1	63	73	11	– Data Input
3						Communications Multiplexer
	98	D8	6F	7F	18	– Controller
	99	D9			19	– Channels
4	83	C3	63	73	03	High Performance Magnetic Tape
5						Wide Range Analog Input System
	92		65	75	12	– Channel Output
	93	D3	66	76	13	– Data Input
6	A0-A7	E0-E7			20-27	Input/Output Interface
7	84	C4	64	74	04	Moderate Performance Magnetic Tape
8	85	C5			05	Card Readers (300 and 1000 CPM)
9	86	C6			06	Card Punch
10	94	D4			14	Wide Range Relay
						Analog Input Subsystem
11	87	C7			07	Line Printer (600 LPM)
12	88	C8			08	X-Y Plotter
13	89	C9			09	Paper Tape Punch
14	8B	CB			0B	Line Printer (50-150 LPM)
15	8A	CA			0A	Teletype/Paper Tape Reader
16	A8-AF	E8-EF			28-2F	Input/Output Interface Subsystem

Table 6-1. Typical Peripheral Device Interrupt Assignments (Non-Virtual Mode)

Each I/O priority interrupt can be connected or disconnected within each peripheral device under program control (See Chapter 8, System Programming). If a peripheral device has a stored interrupt request when a command is issued to disconnect the interrupt, the request will be reset. Interrupt signals are not stored in a disconnected controller, leaving all interrupt signals cleared when a controller is reconnected.

Data Interrupt Sequence

The Level C_{16} data lines provide priority sub-levels for the Data Interrupt. The highest sub-level corresponds to data bit 0 on the data lines and the lowest sub-level to data bit 15.

The data request flip-flop in the device controller is set when one of its peripheral devices requires a data transfer. The setting of any data request, flip-flop will cause the data request line to become true.

If no higher Priority Interrupt level is currently active, the I/O Processor issues a Data Queue Update command, causing all peripheral devices that have their data request flip-flop set to place their priority sub-level on the data lines and their source ID on the source ID lines.

During the Data Queue Update operation each peripheral device examines all of the data lines corresponding to a higher priority sub-level than its own. If a higher priority sub-level is detected, the peripheral device removes its source ID lines.

At the end of the Data Queue Update operation the CPU saves the Source ID of the highest priority peripheral device to define the interrupt entry location. The highest priority peripheral device then resets its data request flip-flop and removes its source ID from the source ID lines. All peripheral devices remove their priority sub-levels from the data lines.

Next, the CPU stores the current contents of the Program Status Doubleword (see Chapter 2) in locations 38_{16}, 59_{16} and branches to the address contained in one of 64 dedicated locations $(80\text{-}BF)_{16}$ specified by the source ID. The entered data transfer subroutine uses the new Program Status word contained in dedicated memory location 58_{16}.

Service Interrupt Sequence

The Service Interrupt operates in the same manner as the Data Interrupt, except that it is connected to level D_{16}, the dedicated entry locations are $(C0\text{-}FF)_{16}$ and the new Program Status word is contained in $5A_{16}$.

INPUT/OUTPUT INSTRUCTIONS

Five instructions are available to initiate, perform and monitor peripheral device I/O operations. See Chapter 8, System Programming for a description of their operation. The first four provide CPU control of all Register I/O operation. The last three are used to perform CPU-independent DMP I/O channel operations. Input/Output Instructions are:

Instruction	*Register I/O*	*DMP I/O*
Output Data	Yes	–
Input Data	Yes	--
Output Command	Yes	Yes
Input Status	Yes	Yes
Initialize DMP	–	Yes

These instructions may be executed in either the Non-Virtual or Virtual Mode, but may not be executed in the Extended Addressing Mode.

REGISTER I/O

Input/output data transfers between peripheral devices and the general purpose registers are handled by the IOP. Commands, data I/O requests, and device status requests are sent to these I/O Processors by the CPU. Register data is transmitted over the 16-bit I/O bus either in an Interrupt Mode or Test and Transfer Mode.

Data Transfer Formats

Register data is transferred over the I/O Bus as a 16-bit word. If a peripheral device requires or generates less than 16 data bits, the device data word occupies the least significant bits of the 16-bit registers and I/O bus with the unused high-order bits appearing as zeroes.

Instruction Execution Sequence

During each Register I/O operation, the device address is transferred from the Instruction Register through an IOP to the addressed peripheral device controller. A set of control signals then are sent to the addressed controller which defines one of the four I/O operations.

If the control signals call for an input, the device places a data word or status word on the 16 data lines of the I/O bus and this word is then transferred to the register specified by the instruction.

If the control signals call for an output, the contents of the specified register are transferred to the Output Buffer Register and then placed on the 16 data lines of the I/O bus.

Interrupt Mode

The Interrupt Mode can be used for Register I/O with any device that generates a transfer request signal. This group of devices includes all standard computer peripherals. The transfer request signal is connected first to an interrupt level. Interrupt service routines perform data transfers and associated control functions.

Test and Transfer Mode

The Test and Transfer Mode is performed for Register I/O by first testing a device by means of the Input Status instruction. When the "Data Ready" status bit equals zero, a transfer can be made to or from the addressed device. The maximum transfer rate in this mode is determined by the timing characteristics of the device and the servicing software.

Direct Memory Processor I/O

In the Non-Virtual Mode of CPU execution, DMP channels address I/O buffer areas located in contiguous Actual Memory. In the Virtual Mode the DMP addresses I/O buffers located in the Virtual addressing space of a user program as defined by a Map Image in memory.

Both modes of CPU execution provide alternative single block or chained block transfer operations.

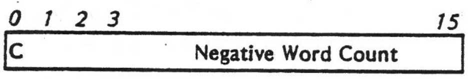
DMP Transfer Parameters

A pair of memory locations is assigned to each DMP peripheral device controller to contain its transfer parameters. The DMP accesses these parameters from the assigned locations and executes them in its channel registers.

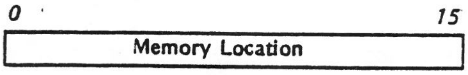
When operating in the Virtual Address Mode, the Transfer Count (TC) and Transfer Address (TA) locations are indirectly accessed in the Operating System addressing space by the virtual address stored in location 60_{16}. This gives access to a relocatable TC/TA table that will accommodate up to 64 unique device addresses.

When operating in the Non-Virtual Address Mode, 16 unique TC/TA pairs are accommodated, the TC parameters at dedicated locations $(60\text{-}6F)_{16}$ and the TA parameters at dedicated locations $(70\text{-}7F)_{16}$.

Transfer Count (TC)



Transfer Address (TA)



Non-Virtual Mode DMP:

TC bit 0: Data chaining (C) Code
0 = Transfer Chain of Blocks
1 = Transfer Single Block

Virtual Mode DMP:

TC bits 0-2: Data Chaining (C) Code

000 = String Chain via O/S Map Image
001 = String Chain via Task Map Image
10X = Transfer Single Block

If the chaining code is 10X, data chaining will not be performed and the TC/TA parameters will have the following meanings:

TC = 10XXXXXXXXXXXXXX, specifying a negative word count of up to 16,384
TA = virtual memory address

If the chaining code is 000, then the data chaining TC/TA parameters will reside in the operating system map (page 1). The less significant TC bits are not used. If the chaining code is 001 then the data chaining TC/TA parameters will reside at an address contained in a map image specified by the DMPI instruction. In either case the TA will equal the virtual address (in page 1 or the page specified by the DMPI) of the individual TC/TA pairs. These TC/TA pairs may be formatted as follows:

TC = 00XXXXXXXXXXXXXX, inferring that more TC/TA pairs follow, or
TC = 10XXXXXXXXXXXXXX, inferring that the present TC/TA pair is the final one. The other 14 bits are a negative transfer count (less than or equal 16,384).
TA = A 16-bit virtual address which utilizes the map image specified by the DMPI.

Non-Virtual Mode DMP

Operation of the DMP in Non-Virtual compatible mode provides connected peripheral devices direct access to contiguously addressed Actual Memory. Both single blocks and chained blocks of up to 16,384 16-bit words each may be transferred. The highest memory address that can be used for I/O buffer areas is 65,535.

Transfer Initiation

Once a DMP peripheral device is appropriately selected and initialized (see Chapter 8, System Programming), a data transfer is started by storing the desired starting address for the transfer in the TA location and the negative number of words to be transferred in the TC location. An Output Command instruction in the Transfer Initiate format is then executed.

Transfer Continuation

Transfers occur automatically at the rate requested by the device. The TA and negative TC are incremented after each transfer.

Transfer Termination

When TC equals zero, a Data Interrupt is generated. If this interrupt is connected by the program to the interrupt (level C_{16}) party line and if the level is enabled, the computer will be interrupted as soon as the data level reaches the top of the interrupt queue.

When the device can accept a new Output Command, the service interrupt (level D_{16}) is generated if the program is connected to the Service Interrupt party line.

The use of both the Data and Service Interrupts provides a choice between two end-of-block signals. One occurs as soon as the last word has been transferred and the other occurs when the device is ready to be commanded again, which often is milliseconds after the last transfer.

Single Block Transfer

If bit 0 of the TC dedicated location is set to one (C=1) before the Transfer Initiate command is executed, only one block of data will be transferred by the selected DMP channel.

Non *Virtual Chain Block Transfer*

If bit 0 of the TC dedicated location is set to zero (C=0) before the Transfer Initiate command is executed, a new block of data words will be transferred automatically after the transfer of the current data block is completed. The data interrupt signifies the completion of each block.

The TC and TA parameters for the new data block are obtained from the two memory locations immediately following those occupied by the current data block. TC is taken from the first location and TA from the second location after the data block. If C=0 in the new TC parameter, data chaining will continue until a TC parameter is encountered with C=1.

Each time a block data transfer is initiated, the contents of the TA and TC dedicated memory locations are automatically transferred to the two registers associated with the channel.

At the end of a block data transfer sequence and just prior to Service Interrupt generation, the final TA automatically is transferred back to the dedicated memory location for the appropriate channel register.

Virtual Mode DMP

When operating in the Virtual Mode the DMP channels transfer single or string-chained data blocks directly to or from I/O buffers located in the virtual addressing space of a selected Task Map Image. The Protect System and Memory Management System automatically are invoked to perform virtual address translation and memory page Access Rights monitoring.

Each data block transferred may contain up to 32K bytes. The addressed I/O Buffer space is limited only by the 128K byte operand map size, which may be defined anywhere in the up to 4 megabytes of available paged memory.

Map Image Selection

During Virtual Mode DMP operations all DMP channel Transfer Addresses (TA's) are defined by a Task Map Image located in memory on an Actual Page boundary. (See figure 6-2.)

The TA location points to an I/O buffer in memory for single block transfers and to a TC/TA parameter list in memory for string chain block transfers. TC/TA parameter lists may be located in the same virtual addressing space that contains the I/O buffers, or they may be located in the virtual addressing space of the Operating System (O/S). The O/S Map Image is located in dedicated Actual Page 1. See figure 6-3.

Selection of the Task Map Image and optional O/S Map Image is accomplished during the Transfer Initiation procedure.

Transfer Initiation

A virtual Mode DMP transfer is begun first by storing the dedicated TC/TA parameters in the O/S addressing space assigned to the required DMP channel and peripheral device controller. TC bits 0-2 specify single-block transfer or string chaining, and the TC/TA parameter list Map Image.

The DMPI instruction is then issued which selects the required DMP channel and loads the channel register with the Actual Page number of a Task Map Image (MIAP) and the number of useable page selection words (PL). (See figure 6-2.)

The Transfer Initiate command is used. Upon going busy, the addressed controller generates a DMP request which causes its DMP channel to initialize, to access the TC and TA parameters stored in the O/S addressing space via the virtual address located at $(60)_{16}$ and to place them in its registers. The channel is now initialized and can handle subsequent DMP data requests. (See Figure 6-4.)

Transfer Continuation

Data transfers occur automatically at the rate requested by the device. The TA and negative TC are incremented after each transfer.

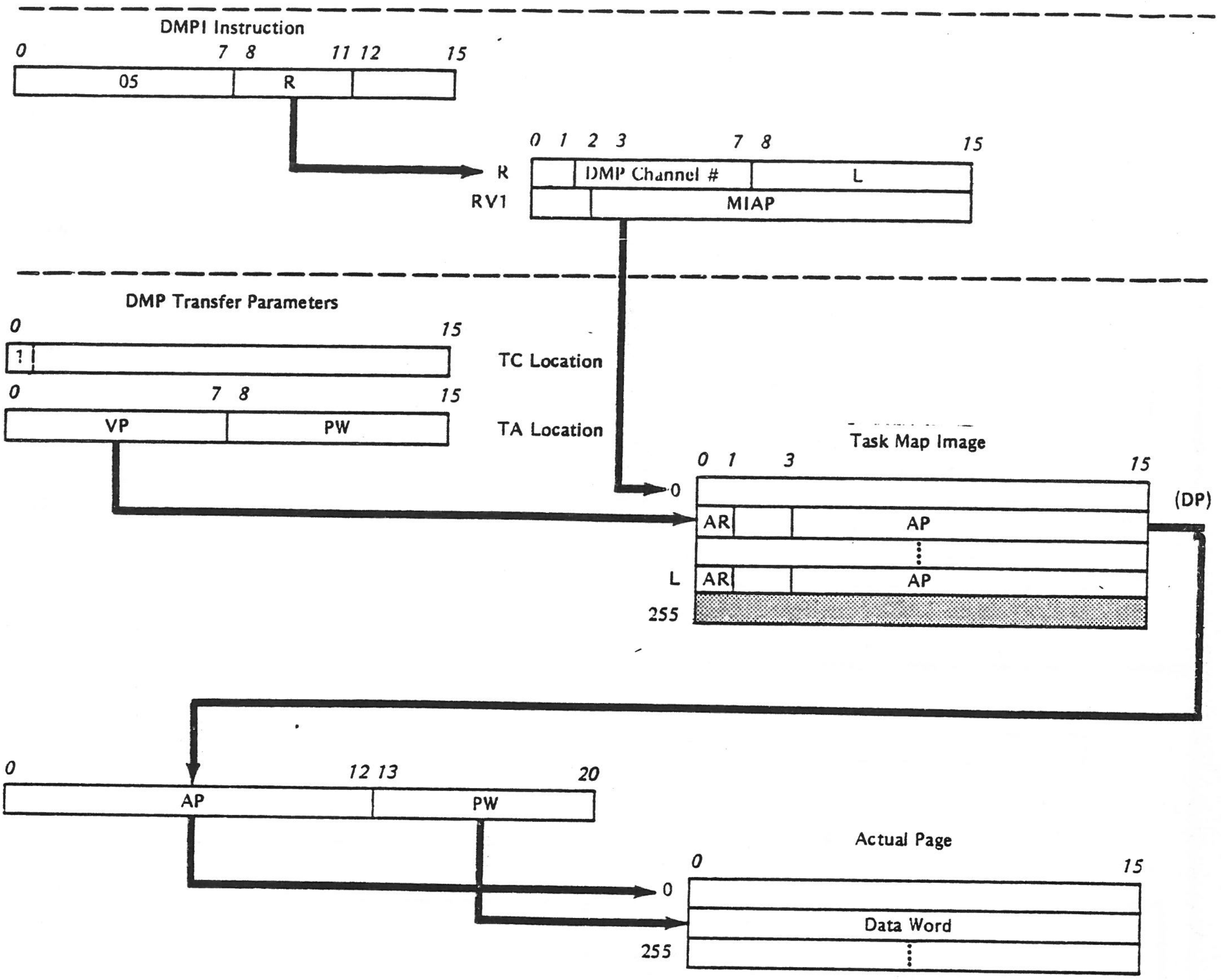
Each time the last data word of an Actual Page is transferred, the TA Page Word (PW) becomes zero, the TA Virtual Page (VP) number is incremented by one which selects a new Actual Page (AP) from the next Task Map Image page selection word, and the block transfer continues until TC becomes zero.

If an attempt is made to access a page selection word outside the useable Page Length (PL) of the Task Map Image, or if the associated page Access Rights (AR) code is set to 00 (no access permitted), the operation is aborted and the DMP channel will issue a Terminate command.

For write operations, if the Access Rights (AR) code is set to 01 (read only) or 10 (read and execute) the operation also is aborted and the DMP channel will issue a Terminate Command.

Transfer Termination

Transfer termination occurs in Virtual Mode DMP operations in the manner described for Non-Virtual Mode DMP operations.



MIAP = Map Image Actual Page : A Memory Page consisting of Actual Page Address accessible by the Task

L = Length Parameter: Usable length of the Task Map Image by the Task

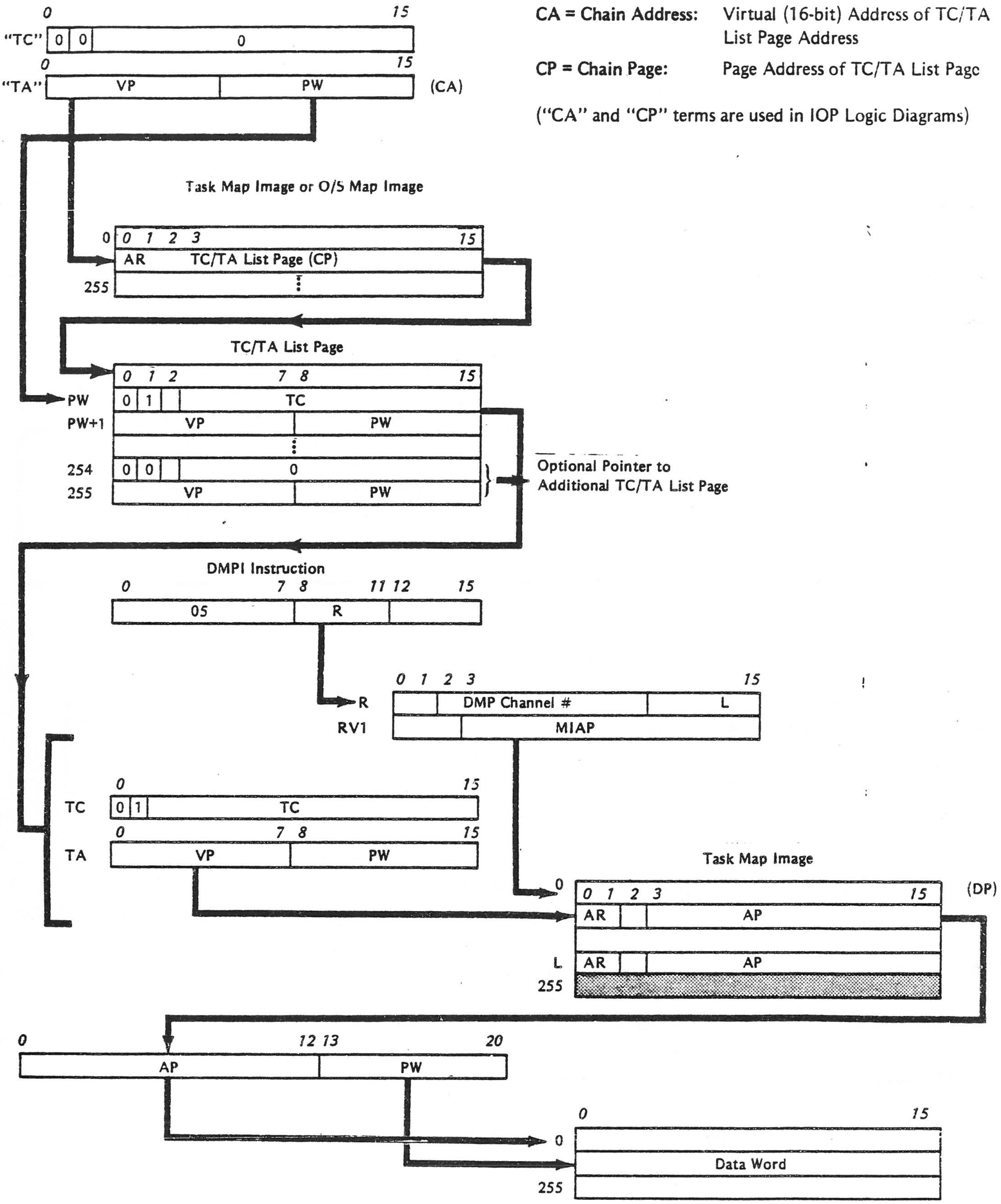
VP = Virtual Page: 8-Bit Address of Memory Location in Task Map Image that contains an Actual Page Address

PW = Page Word: Used to select the Memory Location in the Actual Page where the Data Word is Sourced/Destined

AP = Actual Page: 13-Bit Memory Page Address. Concatenated with PW to form 21-bit Actual Memory Address

DP = Data Page: Another term for AP, used in IOP Logic Diagrams

Figure 6-2. DMP Transfer Address Mapping (No Chaining)



CA = Chain Address: Virtual (16-bit) Address of TC/TA List Page Address

CP = Chain Page: Page Address of TC/TA List Page

("CA" and "CP" terms are used in IOP Logic Diagrams)

Figure 6-3. Virtual Mode String Chaining

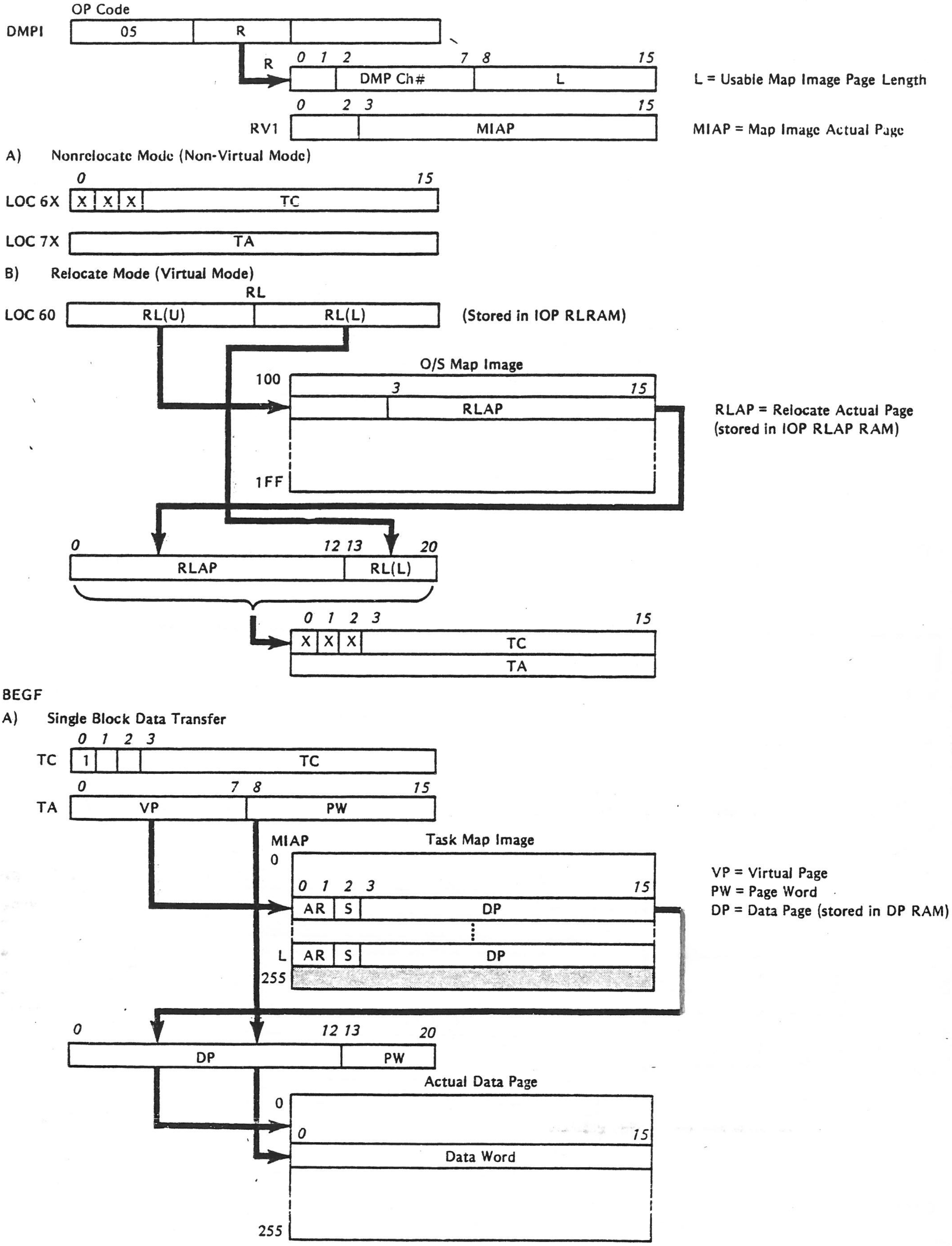
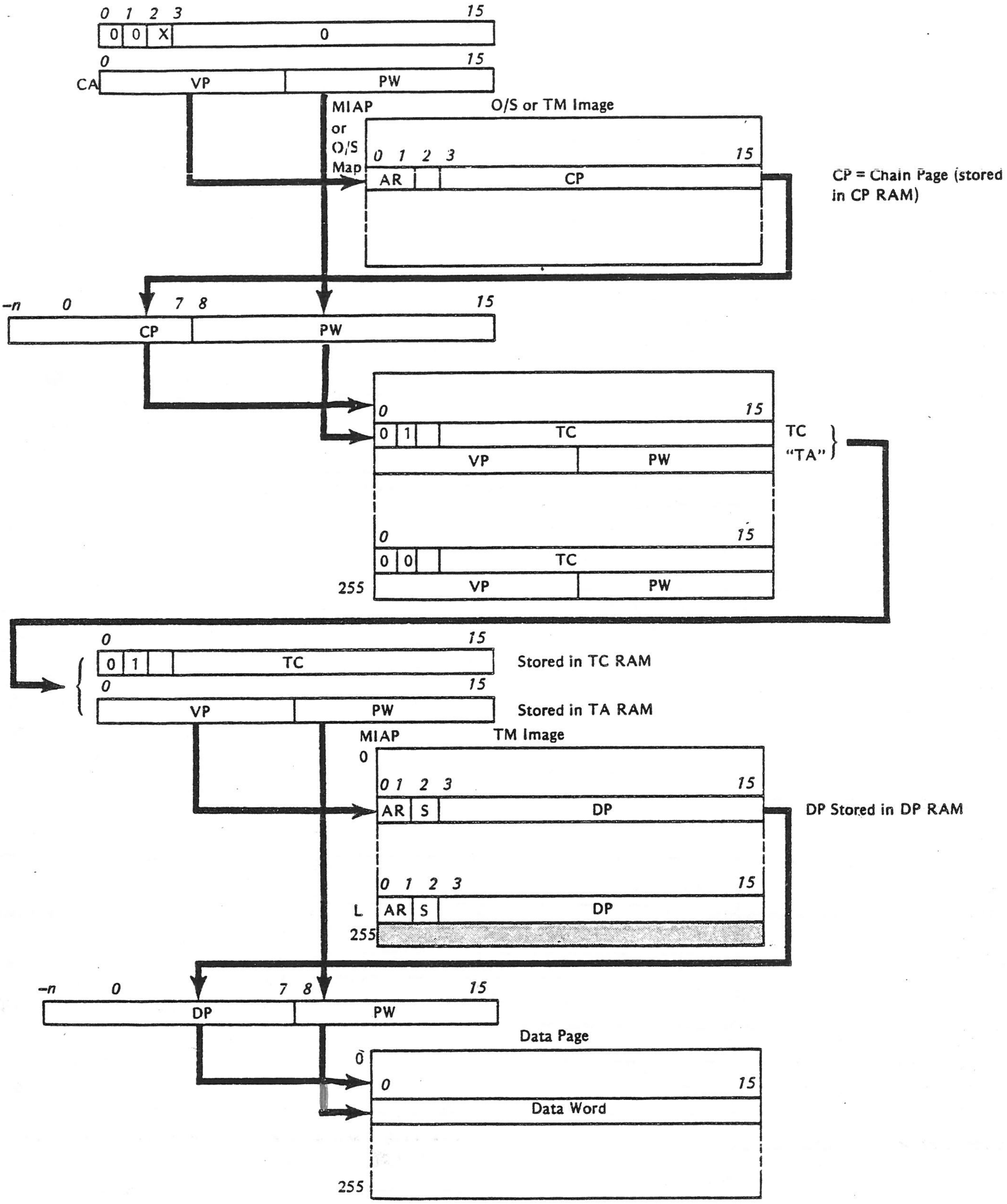


Figure 6-4. MCV Virtual Mode DMP Address Translation (Sheet 1 of 2)

B) String Chaining Data Transfer



CWAF Single block data transfer, and string chaining data transfer
The current TA parameter to location 7X (Non-Relocatable Mode) or to location [RLAP:RL(L)]+1 (Relocatable Mode)

Figure 6-4. MCV Virtual Mode DMP Address Translation (Sheet 2 of 2)

Single Block Transfer

If bit 0 the TC location is set to one (C=1) before the Transfer Initiate command is executed, only one block of data will be transferred by the selected DMP channel (See figure 6-2.)

The TA location contains the Virtual Effective Address of the first data word in the Task Map Image virtual addressing space.

String Chain Block Transfer

If bits 0-2 of the TC location are set to 001 (string chain via Task Map Image) or 000 (string chain via O/S Map Image) before the Transfer Initiate command is issued a chain of data blocks will be transferred by the selected DMP channel using the selected Map Image to define the starting location of TC/TA parameter list. (See figure 6-3.)

The TA location contains the Virtual Effective Address of the TC/TA list.

If bits C-1 of the first TC in the list also are set to 00, the DMP channel will abort the transfer by issuing a Terminate command. Each time TC bits 0-1 are set to 01 the channel uses the next entry (2 locations) in the TC/TA parameter list.

Multiple TC/TA parameter lists may be linked together. Anytime TC bits 0-1 are set to 00 in the transfer parameter list, the DMP channel uses the TA location as the Virtual Effective Address of a new list of TC and TA parameters.

When TC bits 0-1 of the new list also are set to 00, the DMP will issue the normal Terminate Command and enter the Transfer Termination sequence.

I/O Bus Interface

The Classic I/O Bus is a time-shared (party line) bi-directional bus capable of transferring data, commands, and device status. Since the bus is a party line, each device controller must request use of the bus on one or more of the three request lines available. When the bus is available for use, the CPU will issue interlock signals which inform the specified controller when it may use the bus.

The I/O cable is connected serially (daisy-chained) to each Peripheral Controller Interface and Input/Output Interface Subsystem. All controllers and the CPU are connected to the I/O bus through a cable driver/receiver set which isolates the logic from the effect of I/O bus loading and delays.

The I/O cable connection rules are:

- Cable length is switch selectable per I/O bus - either 25 feet total length or 100 feet total length.
- All connections must be made via a cable driver/receiver set.
- A maximum of eight cable driver/receiver sets can be connected to the cable.
- Up to four device controllers can be connected to each driver/receiver set.

I/O Cable Signals

The block diagram of the computer I/O subsystem shows the types of signals in the I/O cable. Thirty-nine pairs of lines are provided for communication between the CPU and external controllers.

Address Signals (6 pairs DA00N through DA05N)

These signals designate which device controller is to respond to the control lines. The six-bit binary value is obtained from the I/O instruction word (bits 6-7, 12-15).

Control Signals (5 pairs)

The four I/O instruction functions are coded on two lines: (DACSN) – Command (false) and data (true), plus (DAOIN) – Input (true) and Output (false).

The I/O Sync signal (IOISN) indicates that an I/O transfer is occurring. When this signal is true, the Command/Data, Input/Output and Address signals are valid.

The other two control signals are Master Clear (IOICB), which is generated at the computer, and Clock (IOCLKN), which is a 5-MHz square wave generated in the computer. IOCLKN is switch-selectable to be either a standard or differential signal. When differential, the IOCLKN return line will be driven as well as IOCLKN. The Master Clear signal normalizes all units and the Clock signal provides a timing signal to controllers, eliminating the need for an internal time base.

Data Signals (16 pairs IOB00N through IOB15N)

All commands, status, data and controller priority are transmitted over this bi-directional bus.

DMP Control Signals (2 pairs)

The DMP Request signal (DMRQN) is sent to the computer by controllers operating under DMP Control. It signifies a request for a word transfer in the direction the DMP channel was last initialized.

The DMP response with a DMP Queue Update signal (UDDMQN). In response to this signal, each (if more than one) controller having a current DMP request places a true signal on one of the 16 data bus lines, indicating one of 16 DMP channel priorities.

The Source ID of the unit, which is a six-bit pointer to the dedicated registers for the channel, is also placed on the Source ID lines. Each requesting unit examines the data bus lines to determine if a higher priority unit is also requesting a transfer. If so, the lower priority unit removes its priority and Source ID signals.

Program Interrupt Control Signals (4 pairs)

These signals consist of a Request signal and Update Queue signal for both of the party line interrupts. The four signals are Data Request (DIRQN), Data Queue Update (UDDIQN), Service Request (SIRQN), and Service Queue Update (UDSIQN). These two pairs of signals are handled just like the DMP Request and DMP Queue Update except they are serviced at priority interrupt levels C_{16} and D_{16}, and DMP Transfers are serviced in the DMP independently of the interrupt priority structure.

Each requesting controller places its priority bit on one of the 16 data bus lines and its Source ID on the Source ID lines. All but the higher priority requesting controller remove their signals during the Update Queue signal before the Source ID signals are transferred to the computer.

Source ID Control Signals (6 pairs SIDOON through SID05N)

Each device has a unique Source ID code which is sent to the computer in response to an update queue command. This code is used to identify the dedicated memory location assigned to the interrupt sub-level or the DMP channel.

Signal Levels

The signal levels at the cable driver/receiver interface to the controller logic are:

Logic 0: 2.4 to 5.5 Volts (Except IOICB, which is inverted)

Logic 1: 0 to 0.4 Volts

The I/O Clock (IOCLKN), a 5-MHz square wave, is distributed on the I/O bus for general purpose use in the controllers.

When the IOICB (Initial Condition Bus) comes true, each controller should normalize (not busy, no interrupt or DMP Requests, no interrupt enabled, device normalized, etc.). This signal is present if the CPU power is going down and is present to normalize the system when power is returned to the CPU and when the MASTER CLEAR switch on the CPU Control Panel is depressed.

All signals are ground true except IOICB. This allows each controller to recognize the absence of a current sink in the CPU and normalize when CPU power is absent.

Chapter 7. CONTROLS AND INDICATORS

General Description

The Classic Control Panel (figure 7-1) contains 42 switches and 68 LED indicators to provide complete control and display of machine operations.

Switches and Indicators

The Controls and Indicators are classified as:

Switches

- Execution Control
- Register Display Select
- Memory Mode
- Enter/Display Control
- Switch Register

Indicators

- Machine Status
- Address
- Data
- Switch Register

Execution Control Switches

Execution Control switches include the Enable/Disable keylock switch, the Master Clear, Fill, Halt/Run and Single Step switches. They are shaded for illustration in figure 7-2.

Enable/Disable

In the ENABLE position, all Control Panel switches are enabled. In the DISABLE position, all Control Panel switches except for the Register Display, Switch Register and Console Interrupt switches are disabled.

M Clear

Depressing the M CLEAR (Master Clear) switch normalizes the entire computer system. The CPU control logic is reset, peripheral I/O controllers are normalized and the PR register is set to zero. If depressed while the computer is in the RUN state the computer will return to the RUN state after the Master Clear (firmware) algorithm is completed. An additional feature is the maintenance clear function, which is activated by depressing the Console Interrupt (CSL INT) simultaneously with the M CLEAR. This causes a similar clearing algorithm, but leaves certain internal status conditions, condition codes, interrupt and address registers intact to facilitate maintenance activity.

Fill

Depressing the FILL switch causes a bootstrap routine to be transferred from permanent storage within the CPU plane to memory. Fill device is determined by the state of the switch register switches. Fill is disabled when in the RUN state.

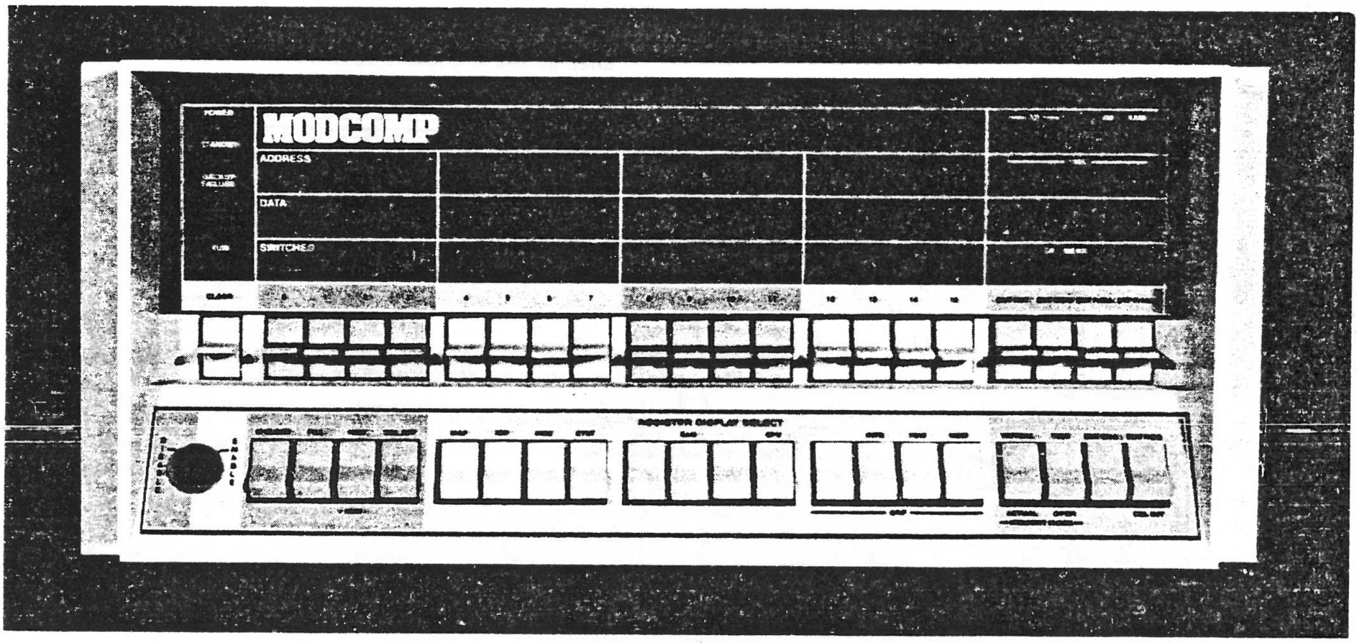


Figure 7-1. Classic Control Panel

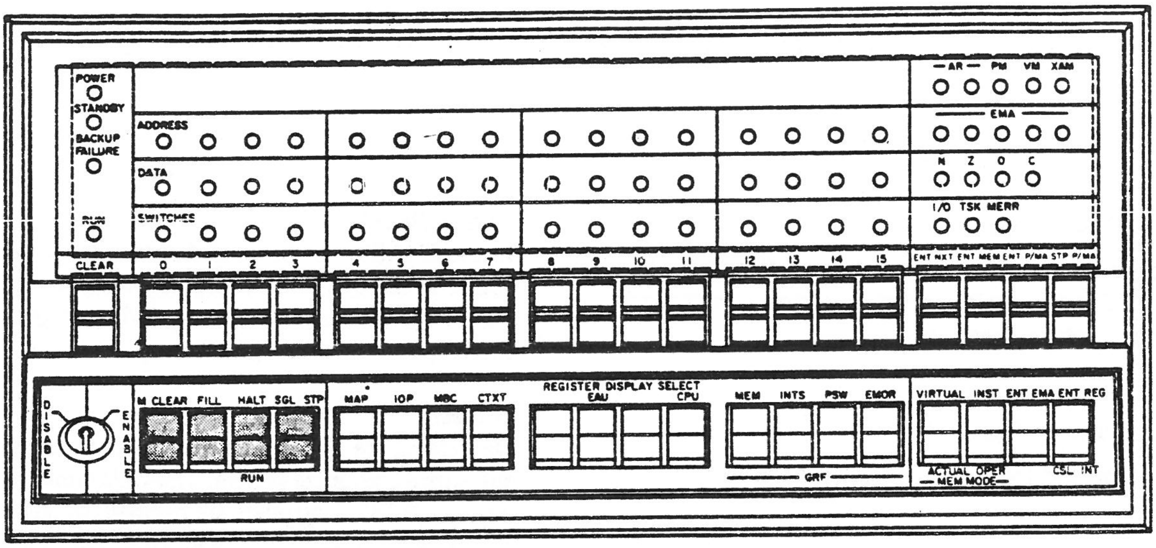


Figure 7-2. Execution Control Switches (shaded)

Run/Halt

The RUN/HALT switch controls program execution. In the HALT position, the CPU cycles within a closed-loop control panel service algorithm (firmware). This enables the various enter and display functions. When depressed to the RUN position, the CPU begins program execution at the memory location stored in the program counter. Program execution stops when the RUN/HALT switch is returned to the HALT position.

SGL STP

Depressing the Single Step (SGL STP) switch causes the CPU to execute the instruction beginning at the address specified by the program register. Upon completing the instruction, the CPU will return to the HALT state. The SGL STP switch is disabled during the RUN state.

Register Display Switches

The twelve REGISTER DISPLAY SELECT switches illustrated in figure 7-3 select the internal computer register or other storage location whose contents are displayed by the 16 DATA display LEDs. The selected location may, in most cases, be entered using the Enter/Display Control switches. The internal computer registers selectable by the REGISTER DISPLAY SELECT switches are tabulated in figure 7-4. When the computer is halted, the internal register selected by these switches is continuously displayed on the DATA display LEDs. The various enter register and memory features of the Control Panel are then available. When the CPU is in the RUN state, the DATA display is automatically updated with the contents of the selected internal register upon completion of most instructions. This capability provides a dynamic display of the many selectable registers at no cost in performance to the executing software.

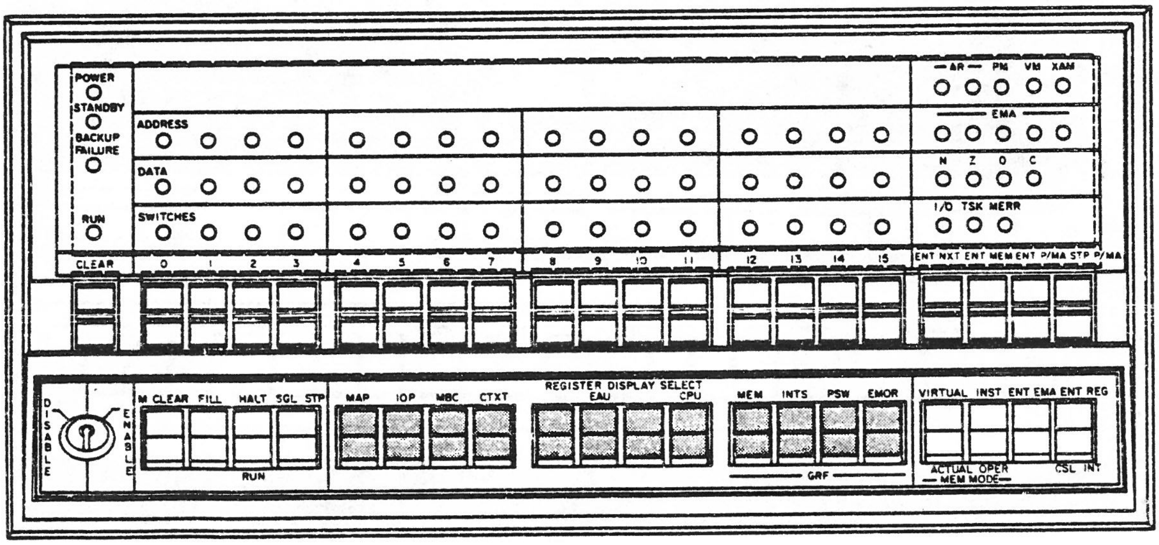


Figure 7-3. Register Display Select Switches (shaded)

Switch Setting	*Will Display*	*Will Select*	*On Plane*
000-00F	GRF	GRF	CPU
010	MAP RAM INPUT	MAP RAM OUTPUT	
011	EMCR	EMCR	
012	PSW	PSW	
013	EMA	EMA	
014	ACT	ACT	
015	REQ	REQ	
016	ENA	ENA	
017	MA	(NONE)	
018	OS	(NONE)	
019	IS	(NONE)	
01A	IR	(NONE)	
01B	PTB (PR)	(NONE)	
01C	LIT/AG	(NONE)	
01D	TW	(NONE)	
01E	DSW	(NONE)	
01F	MDB	(NONE)	
041-04F	GRF COPY 0		EAU
051-05F	GRF COPY 1		
061-06F	GRF COPY 2		
071-07F	GRF COPY 3		
100-1FF	CONTEXT FILES		
400-4FF*	BUS 1 DMP DATA AND ADDRESS PARAMETERS		IOP 1
500-5FF*	BUS 2 DMP DATA AND ADDRESS PARAMETERS		
600-6FF*	BUS 1 DMP DATA AND ADDRESS PARAMETERS		IOP 2
700-7FF*	BUS 2 DMP DATA AND ADDRESS PARAMETERS		
800-8FF**	VIRTUAL MAPPING RAMS		CPU

*FOR IOP REGISTERS SELECTED BY 4XY, 5XY, 6XY, 7XY; Y = DMP Channel Number, and X = Register Number as shown below.

0: OUTPUT WORD (OW)
1: TRANSFER COUNT (TC)
2: TRANSFER ADDRESS (TA)
3: CHAIN ADDRESS (CA)
4: RELOCATE PARAMETER (RL)
5: OUTPUT TRANSFER COUNT (OTC)
6: OUTPUT TRANSFER ADDRESS (OTA)
7: OUTPUT CHAIN ADDRESS (OCA)
8: OUTPUT WORD (OW)
9: DATA PAGE (DP)
A: CHAIN PAGE (CP)
B: UNUSED
C: RLU/RLAP: UPPER BYTE OF RELOCATE PARAMETER
D: OUTPUT DATA PAGE (ODP)
E: OUTPUT CHAIN PAGE (OCP)
F: TASK MAP (TW)

** For CPU Virtual Mapping Rams selected by 800-8FF, Individual Map location is selected as follows:

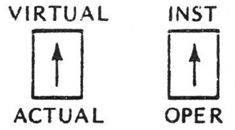
1: M0 M1 M2:R0 R1 R2 R3 R4 R5 R6 R7
M0–2 = MAP 1-8 SELECTION
R0–7 = VIRTUAL PAGE REGISTER

Figure 7-4. Register Display Select

Memory Mode Switches

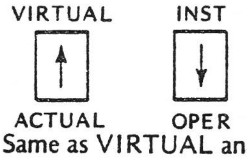
The two Memory/Mode switches determine the addressing mode used during control panel enter and display memory functions. (Refer to Enter/Display Control). They are illustrated in Figure 7-5. The four possible unique settings of those switches are defined below.

VIRTUAL and INST



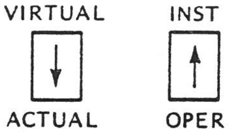
If the CPU is in the Non-Virtual mode, the INST switch has no effect and memory addressing from the control panel is performed in the Non-Virtual mode. If the CPU is in the virtual mode, memory addressing from the control panel is virtual, via the instruction map currently selected by the Program Status Word.

VIRTUAL and OPER



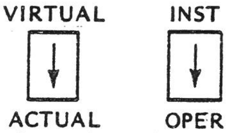
Same as VIRTUAL and INST except that if the CPU is in the virtual mode, control panel memory addressing occurs via the currently selected operand map.

ACTUAL and INST



Control panel memory addressing occurs in the actual (i.e., physical) mode. In this mode the 21-bit address formed by EMA:MA registers is utilized to access memory. Virtual mapping RAMs are bypassed regardless of current CPU addressing mode.

ACTUAL and OPER



Control panel memory addressing occurs in the extended mode. In this mode EMA:MA registers are added to a modulo-8K lower boundary value stored in the CPU Extended Memory Control Register (EMCR). The result is verified to be below a modulo-8K upper boundary also stored in the EMCR. If so, the 21-bit result is used to access memory. Virtual mapping RAMs are bypassed regardless of CPU addressing mode.

Note. These switches do not affect instruction or operand addressing by the computer itself - only the mode of memory access by the control panel.

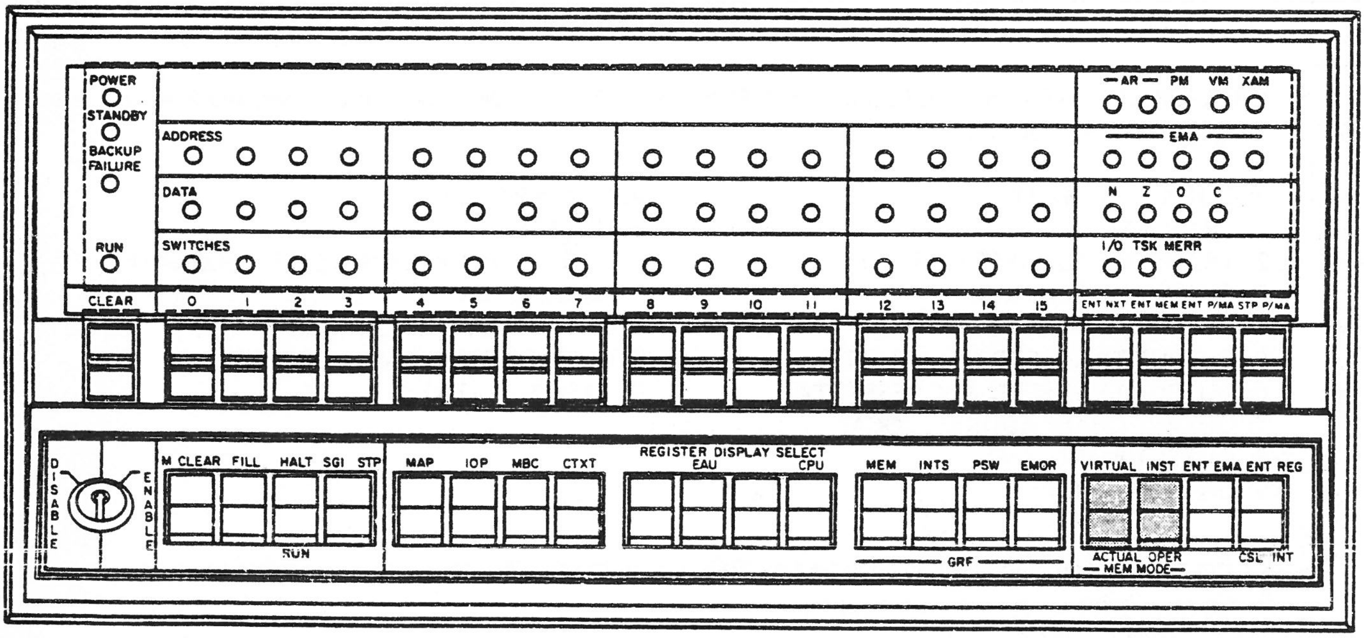


Figure 7-5. Memory Mode Switches (shaded)

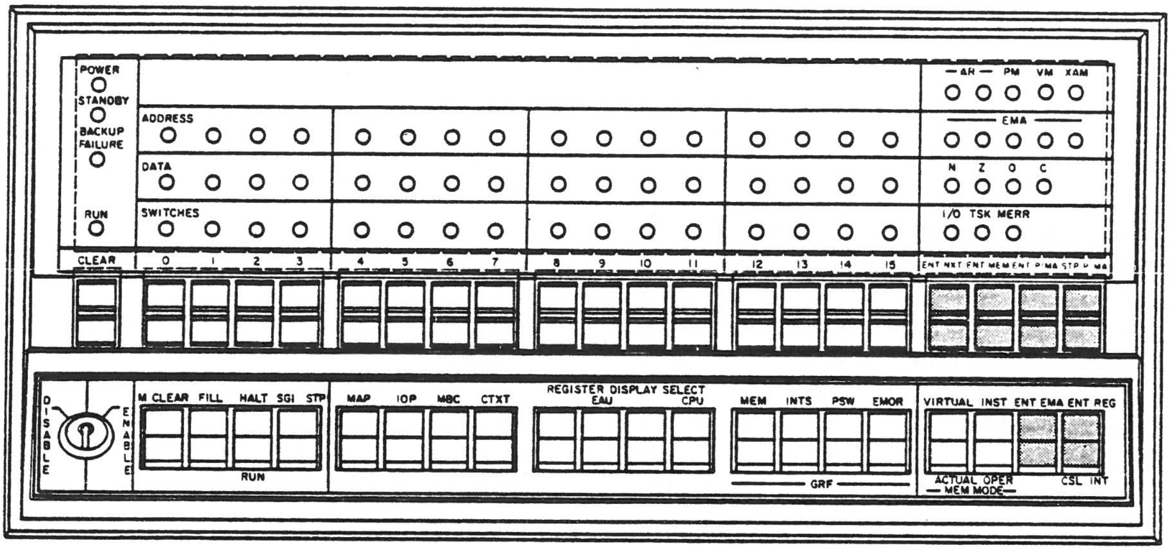


Figure 7-6. Enter/Display Control Switches (shaded)

Enter/Display Control Switches

The following switches perform Enter and Display functions within the Classic. They are illustrated in figure 7-6.

- ENT EMA
- ENT REG/CSL INT
- ENT P/MA
- STP P/MA
- ENT MEM
- ENT NXT

ENT EMA, (Enter EMA) when depressed, loads control panel data switch bits 11-15 into the CPU Extended Memory Address Register. This function is only enabled when the computer is halted.

ENT REG/CSL INT

The ENT REG/CSL INT (Enter Register/Console Interrupt) switch performs two functions. When the CPU is halted, depression of this switch causes the internal register determined by Register Display Select switch settings to be loaded with the value of the Data Switches. When the CPU is running, the enter register function is inhibited and this switch will generate a level E interrupt request when depressed. The Classic reserves this interrupt level for a console interrupt.

ENT P/MA

Depressing the ENT P/MA (Enter P/MA) switch loads the CPU Program Register (PR) and/or Memory Address Register (MAR). The PR is reserved for instruction Memory addresses, while the MA register is used to store operand memory addresses. When the two Memory Mode switches are in the VIRTUAL and INST position, Enter P/MA will destine the data switch value to the PR and also to the MA. All other Memory Mode settings will cause only the MA register to be loaded. In all cases, the new MAR contents will cause a memory call to be made using the selected addressing mode; the resultant read data will be destined to the control panel data indicators. This switch is only enabled when the computer is halted.

STP P/MA

Depressing the STP P/MA (Step P/MA) switch causes the PR and/or MAR to be incremented. Memory Mode switch-setting of VIRTUAL and INST will cause the PR to be incremented and the new address loaded into the MAR as well. For all other Memory Mode switch settings, the MAR is incremented. In all cases the new MAR contents will cause the CPU to read the new memory location via the selected mode, and display the contents on the DATA display LEDs. This switch is only enabled when the computer is halted.

ENT MEM

Deoressing the ENT MEM (Enter Memory) switch causes the DATA switch value to be written into the memory address stored in the MAR. Memory is addressed via the selected control panel Memory Mode. This switch is only enabled when the computer is halted.

ENT NXT

ENT NXT (Enter Next), combines the functions of Step P/MA and Enter Memory in a single switch to facilitate sequential memory entry. When depressed, Enter Next causes the MAR contents to be incremented and then writes the DATA switch value into the addressed location. This switch is only enabled when the computer is halted.

Switch Register Switches

The Data SWITCHES and the Switch Register CLEAR switch are 3-position switches-momentary when depressed downward from the center position and latching when raised upward from the center position. The 16 data switches serve as inputs to the Control Panel Switch Register. When either pressed downward (momentary action) or raised upward (latching) the corresponding bit will be set in the Switch Register and immediately displayed on the switch indicators. The Switch Register CLEAR switch manually resets the switch register contents when depressed. When in the center position only Enter Memory, Enter Register, Enter EMA, Enter Next or Enter P/MA operation will automatically reset the Switch Register. Latching the CLEAR switch in the upward position inhibits this automatic clear feature, so that only a Master Clear or depression of the CLEAR switch will reset the register. These switches are illustrated in figure 7-7.

Machine Status Indicators

The 16 LED indicators illustrated in figure 7-8 display the status of computer operation, its current modes of operation and basic power status.

POWER

When lit indicates that power supply output voltages are within operable limits.

STANDBY

Indicates that battery back-up unit is charged and armed.

BACK-UP FAILURE

Discharged battery in battery back-up unit. This indicator will be lit upon power-up after a power-fail that lasted longer than the back-up unit could support.

RUN

Indicates that computer is in RUN state.

AR

These two indicators display the access rights of the current Virtual Address in the PR or MAR. These indicators are only relevant when the CPU is performing Virtual Mode Memory addressing.

VM

Indicates when the Classic CPU is in the Virtual Mode of operation.

PM

Indicates the Classic CPU is in the piplined mode of operation.

XAM

Indicates when the CPU is performing an operand in the extended addressing mode.

N,Z,O,C

These four indicators display the current states of the four condition codes - Negative, Zero, Overflow, and Carry.

I/O

Indicates that interrupt level C or D is active. These are the two interrupt levels specifically reserved to service Input/Output devices.

TSK

Indicates that interrupt level F, taskmaster interrupt level is activated.

MERR

Indicates that interrupt level 1, reserved for memory error handling, is active.

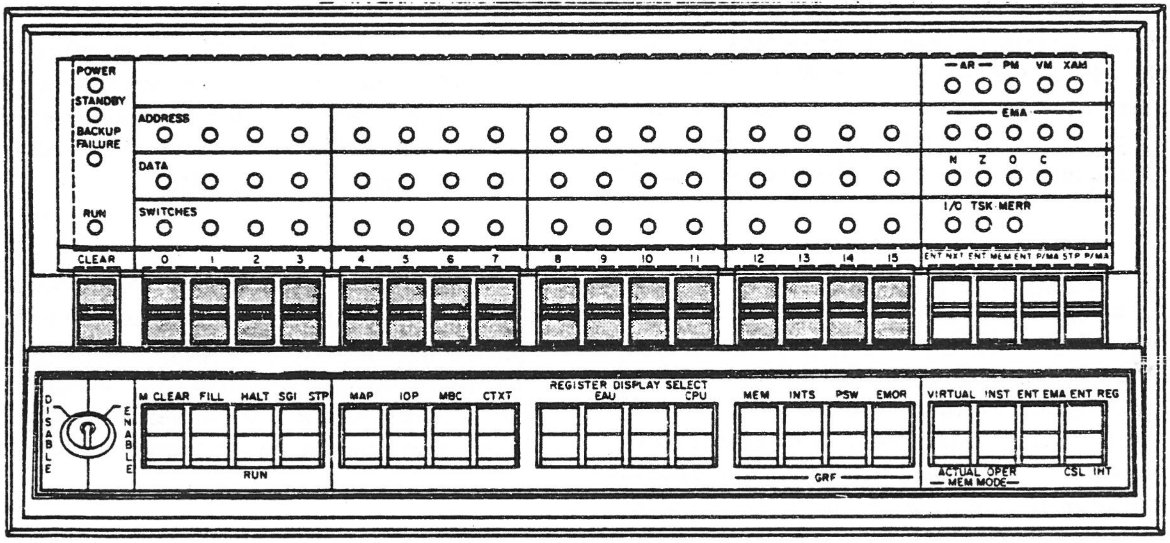


Figure 7-7. Switch Register Switches (shaded)

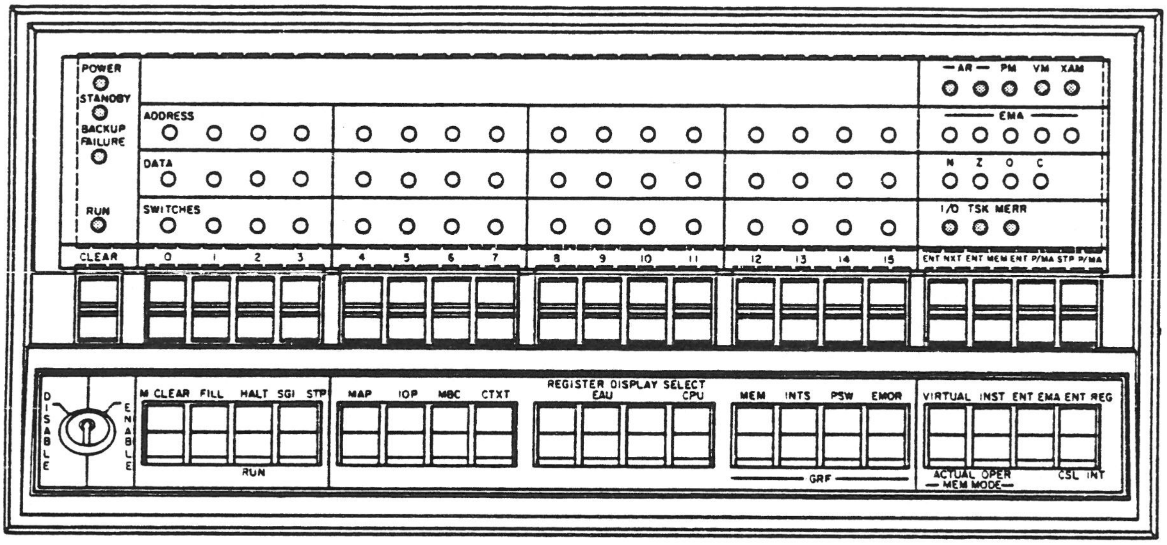


Figure 7-8. Machine Status Indicators (shaded)

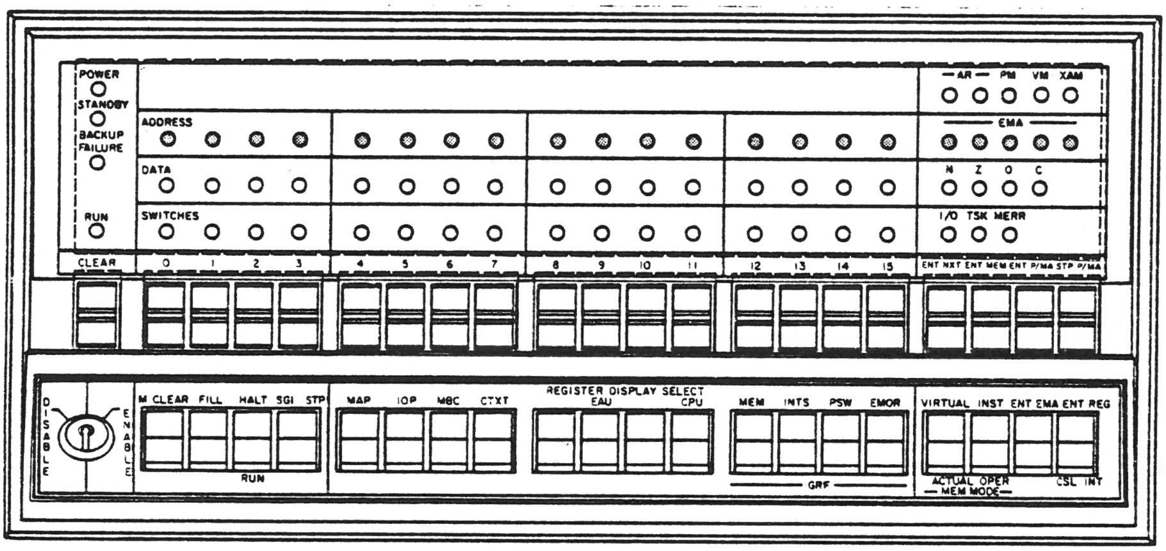


Figure 7-9. Address Display (shaded)

Address Indicators

The control panel displays 21 bits of address information, to accommodate the 2MW memory addressability of the Classic. These address indicators display the current contents of the MA register.

Chapter 8. SYSTEM PROGRAMMING

Programming Overview

The Classic processes operands in the General Purpose Register File, Memory, Address Mapping Files, Register Context File, and in Hardware Status Registers.

Operands

Operands may be bits, bytes, words (16-bits), double-words (32-bits), triple words (48-bits), quadruple-words (64-bits), files of from one to eight words and push/pull stacks of any number of words, or memory blocks of any number of words.

Bit addressing may access any one of 16 bits per word, either in a register or in memory. Byte addressing may access either of two 8-bit bytes per word in registers or in a register and memory.

Word and multi-word addressing access file operands in registers or in registers and memory. Stack addressing allocates and deallocates space in memory stacks and transfers up to 16 operands between registers and memory.

Memory Addressing

Memory addressing may be any combination of direct, indexed or indirect (to one level), or it may be base displaced, program counter relative or immediate. The last two types of addressing refer to the program's current (virtual) instruction space; the others refer to the program's current (virtual) operand space. Many instructions may also access operands in extended memory as defined by the boundary values in the Extended Memory Control Register (EMCR).

Address Map Files

The instruction space and operand space may be the same or may differ, depending on the Address Mapping File(s) selected. An Address Mapping File (Map) may define up to 32K or 128K bytes of addressing space, depending upon which Map is selected. One or more programs may reside (be mapped) in the same instruction space and share data areas in the same operand space.

When the mode of CPU execution is Non-Virtual, all programs and their data reside in and share the same actual memory space.

When the mode of CPU execution is Extended addressing, all memory not between that defined by the Extended Memory Control Register Upper and Lower Boundaries (set in increments of 8K, from 0 to 2 megawords) is protected and may not be accessed by any Extended Addressing Mode instruction.

Instructions

Instructions that may use Extended Addressing are defined later in this chapter.

Classic instructions include the basic MODCOMP II compatible instruction set, with the following powerful additions: 32-bit fixed point, 64-bit floating point, stack push/pull, paged memory management, and miscellaneous software efficiency instructions, and additional instructions designed to enhance FORTRAN execution.

Equipped with this instruction set, the Classic performs load, store, and transfer operations; fixed point and floating point arithmetic; logical, shift, compare, test, branch, and hop functions; CPU control; interrupt processing; and input/output operations.

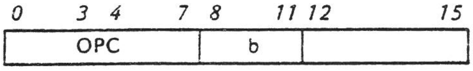
Many of the Classic instructions are privileged and several will operate only in the Virtual Mode of execution. Most Classic instructions will set condition codes to be sensed by the Conditional Branch instructions.

Each Classic instruction operation code (OPC) consists of eight control bits that uniquely select the length of its operands in the general registers and in memory, applicable memory operand addressing mode and specific function the instruction will perform. Some instructions contain one or two operation augment codes which assist in defining these parameters.

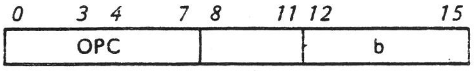
Bit Constants/Addressing

Bit constants (or powers-of-two) are used in many Classic instructions to manipulate bit operands in general registers or in memory.

In some contexts (e.g., logical operations) bit constants are one-bit quantities of value 1 in any of the bit positions (0-15) of a 16-bit word. Such bit constants are positive powers-of-two. In other contexts (e.g., arithmetic operations) bit constants are full 16-bit quantities of value ± (15-b), where b is the bit position (0-15). These bit constants are negative powers-of-two and are used as bit field masks. This bit position designator (b) is a four-bit, binary-coded, bit field in the instruction.



Addressing a bit in memory



Addressing a bit in a register

Figure 8-1 and 8-2 show all of the formats and values that may be specified for bit constants as they are defined for use by instructions that perform bit processing.

Bit (b)	Ø							7	8							15	Hexadecimal	2^{15-b}	Decimal
Ø	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	8ØØØ	2^{15}	-32768
1	Ø	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	4ØØØ	2^{14}	16384
2	Ø	Ø	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	2ØØØ	2^{13}	8192
3	Ø	Ø	Ø	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1ØØØ	2^{12}	4096
4	Ø	Ø	Ø	Ø	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø8ØØ	2^{11}	2048
5	Ø	Ø	Ø	Ø	Ø	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø4ØØ	2^{10}	1024
6	Ø	Ø	Ø	Ø	Ø	Ø	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø2ØØ	2^{9}	512
7	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø1ØØ	2^{8}	256
8	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	ØØ8Ø	2^{7}	128
9	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1	Ø	Ø	Ø	Ø	Ø	Ø	ØØ4Ø	2^{6}	64
1Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1	Ø	Ø	Ø	Ø	Ø	ØØ2Ø	2^{5}	32
11	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1	Ø	Ø	Ø	Ø	ØØ1Ø	2^{4}	16
12	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1	Ø	Ø	Ø	ØØØ8	2^{3}	8
13	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1	Ø	Ø	ØØØ4	2^{2}	4
14	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1	Ø	ØØØ2	2^{1}	2
15	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1	ØØØ1	2^{0}	1

Figure 8-1. Bit Constants as Powers-of-Two

Bit (b)	Ø							7	8							15	Hexadecimal	2^{15-b}	Decimal
Ø	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	8ØØØ	-2^{15}	-32768
1	1	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	CØØØ	-2^{14}	-16384
2	1	1	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	EØØØ	-2^{13}	-8192
3	1	1	1	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	FØØØ	-2^{12}	-4096
4	1	1	1	1	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	F8CØ	-2^{11}	-2048
5	1	1	1	1	1	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	FCØØ	-2^{10}	-1024
6	1	1	1	1	1	1	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	FEØØ	-2^{9}	-512
7	1	1	1	1	1	1	1	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	FFØØ	-2^{8}	-256
8	1	1	1	1	1	1	1	1	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	FF8Ø	-2^{7}	-128
9	1	1	1	1	1	1	1	1	1	1	Ø	Ø	Ø	Ø	Ø	Ø	FFCØ	-2^{6}	-64
1Ø	1	1	1	1	1	1	1	1	1	1	1	Ø	Ø	Ø	Ø	Ø	FFEØ	-2^{5}	-32
11	1	1	1	1	1	1	1	1	1	1	1	1	Ø	Ø	Ø	Ø	FFFØ	-2^{4}	-16
12	1	1	1	1	1	1	1	1	1	1	1	1	1	Ø	Ø	Ø	FFF8	-2^{3}	-8
13	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Ø	Ø	FFFC	-2^{2}	-4
14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	Ø	FFFE	-2^{1}	-2
15	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FFFF	-2^{0}	-1

Figure 8-2. Bit Constants as Negative Powers-of-Two

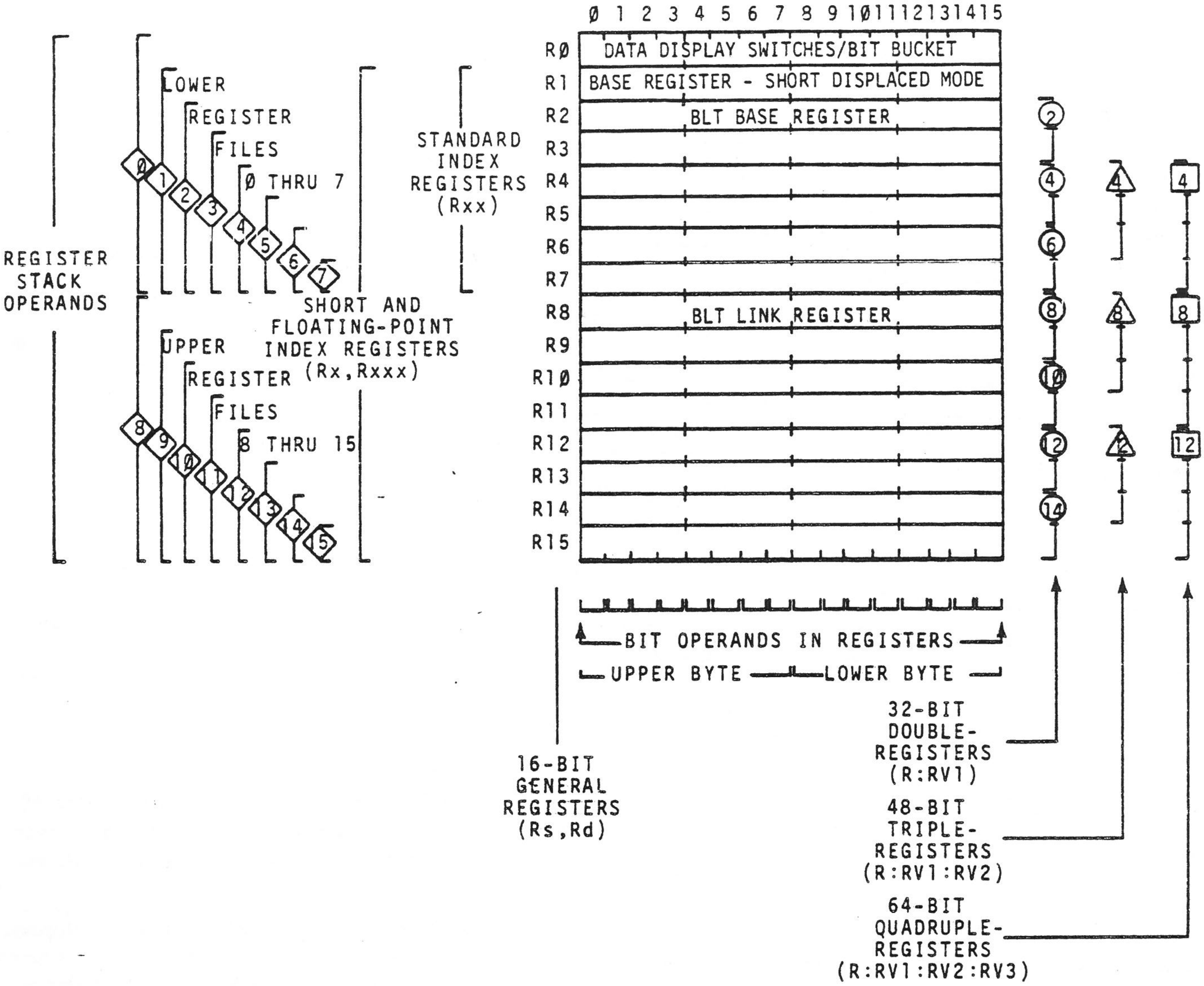
Register Operands/Addressing

Individual or multiple registers in the General Purpose Register File may contain either destination or source operands.

Register File Organization

Fifteen 16-bit hardware registers (see figure 8-3) are available to the programmer, and may be manipulated by most machine instructions. Register 0 (R0) contains the Control Panel DATA DISPLAY switches when sourced, and functions as a non-storage "bit-bucket" when destined. Register 1 (R1) thru register 15 (R15) are general in use, configuration and operation, with registers 1 thru 7 also designated to be used as index registers for standard indexed addressing and as extended address registers for extended addressing.

The general registers may be addressed in a variety of ways. The programmer may configure them according to his needs, into appropriate combination s of 16-bit, 32-bit, 48-bit or 64-bit accumulators. Operands in the general registers may be manipulated with each other, with bit constants, or with operands in memory.

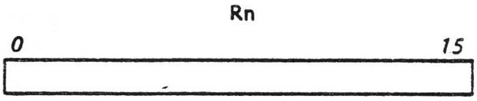


Ø 1 2 3 4 5 6 7 8 9 1Ø 11 12 13 14 15
RØ DATA DISPLAY SWITCHES/BIT BUCKET
R1 BASE REGISTER - SHORT DISPLACED MODE
R2 BLT BASE REGISTER
R3
R4
R5
R6
R7
R8 BLT LINK REGISTER
R9
R1Ø
R11
R12
R13
R14
R15
REGISTER STACK OPERANDS
LOWER REGISTER FILES Ø THRU 7
UPPER REGISTER FILES 8 THRU 15
STANDARD INDEX REGISTERS (Rxx)
SHORT AND FLOATING-POINT INDEX REGISTERS (Rx,Rxxx)
Ø
1
2
3
4
5
6
7
8
9
1Ø
11
12
13
14
15
BIT OPERANDS IN REGISTERS
UPPER BYTE
LOWER BYTE
16-BIT GENERAL REGISTERS (Rs,Rd)
32-BIT DOUBLE-REGISTERS (R:RV1)
48-BIT TRIPLE-REGISTERS (R:RV1:RV2)
64-BIT QUADRUPLE-REGISTERS (R:RV1:RV2:RV3)
2
4
6
8
1Ø
12
14

Figure 8-3

Single-Register Operands

Each general register may be manipulated separately.

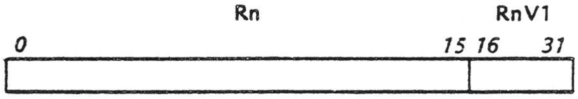


Single-registers are designated as Rn. RO has the special function of containing DATA DISPLAY switch values and R1 has a secondary special function as an implied base register for short-displaced addressing. R1 through R15 are general in purpose. R1 through R7 may be used for indexing and R1 through R15 may be used for short instruction indexing and to index floating point operations.

Each single-register may contain one 16-bit fixed point, logical or address value, two 8-bit byte values, or sixteen 1-bit binary values. Sets of contiguous single-registers are addressable as a register file or stack.

Double-Register Operands

The general registers may be manipulated in concatenated even/odd register pairs, starting with R2.



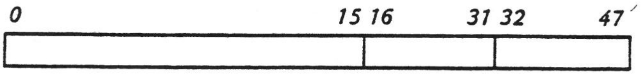
Double-registers are designated Rn:RnV1. Each double-register may contain one 32-bit fixed point value or one 32-bit floating-point value.

In addition, R2 through R6 may be used for extended operand addressing of indexed instructions, and R2-R14 may be used for extended operand addressing of short-indexed and indexed Floating Point instructions.

A double-register cannot be formed from a pair of registers whose initial address is odd, since the second register address is computed by logically OR'ing (V) a hex one with the 4-bit address of the initial register. Should the initial register address be odd, the computed extension register would be the same as the initial register.

Triple-Register Operands

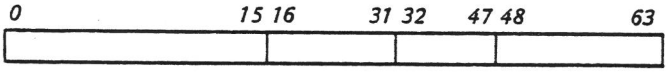
The general registers may be manipulated in concatenated register triples, starting with R4, R8 or R12.



Triple-registers are designated Rn:RnV1:RnV2. Each triple-register may contain one 48-bit floating point value.

Quadruple-Register Operands

The general registers may be manipulated in concatenated register quadruples, starting with R4, R8 or R12.



Quadruple-registers are designated Rn:RnV1:RnV2:RnV3. Each quadruple-register may contain one 64-bit floating point value, or the extended results of some 32-bit fixed-point operations.

Bit Operands

Individual bits in the general registers may be manipulated either logically or arithmetically. There are 16 separate bits in each register.

Byte Operands

Individual bytes may be moved and interchanged in the general registers and may be transferred between the general registers and memory. There are two separate 8-bit bytes in each register.

Register File Operands

Operands in two separate sets of from one to eight general registers may be transferred between the registers and memory as files. Included are the set of registers R0 through R7 (called the lower file) and R8 through R15 (called the upper file). All successively higher numbered registers from the referenced register through the file-end register (R7 or R15) are transferred.

Register Stack Operands

Operands in from 1 to 16 general registers may be transferred between the registers and last-in-first-out (LIFO) memory stacks. Any register may be the beginning register. Successively higher numbered registers are transferred, including wrap-around from R15 through R0, as determined by the register count.

Register Addressing

Single-register and multiple-register operands are addressed by one or two 4-bit fields in the first word of an instruction. Operands in registers may be source or destination operands. Register-to-register instructions use both operand types. Bit-in-register instructions use a destination register and designate a specified bit. Register-memory instructions use a source or destination register and an optional memory operand index register designator.

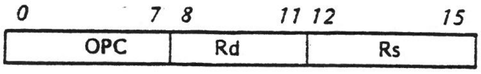
Single-Register Addressing

Byte, word, file or stack operands in single source and/or destination register(s) are addressed by standard 4-bit register designation field(s) in the instruction. Bit operands in single destination registers are addressed by a separate 4-bit designation field in the instruction. The register designation field also is used to address single registers used as an index, register in memory operand addressing.

For all instructions in which the contents of a source register (Rs), either with or without manipulation, are transferred to a destination register (Rd) the two register addresses may be made the same to produce a single-register operation.

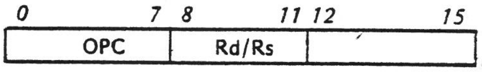
Standard Addressing

The following two instruction formats are used for source and/or destination single-register addressing of byte, word, file or stack operands.



(Source and destination registers)

Bits 12-15 of the instruction specify the source register (0-15) operand and bits 8-11 specify the destination register (0-15) operand in register-to-register operations.

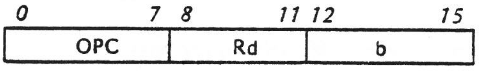


(Source or destination register)

Bits 8-11 of the instruction specify either the source register or destination register operand in register-memory operation.

Bit Addressing

The following instruction format is used for destination bit-in register operand addressing.



(Bit in destination register)

Bits 8-11 of the instruction specify the destination register (0-15) containing 16 separately addressable bits. Bits 12-15 of the instruction word specify the desired bit position (0-15) in the designated register.

Index Addressing

The following three instruction formats are used for index register designation in memory operand addressing (Memory operand addressing is described later in this section).

0 7	8 11	12 15
OPC		Rxxx

(Index register - floating point)

Bits 12-15 of the instruction specify the index register (1-15) in floating point indexed register-memory operations.

0 7	8 11	12 15
OPC		Rx

(Index register - short instructions)

Bits 12-15 of the instruction specify the index register (1-15) in short instruction indexed register-memory operations.

0 7	8 11	12	15
OPC		I	Rxx

(Index register - standard)

Bits 13-15 of the instruction word specify the index register (1-7) in standard direct/indexed/indirect register-memory operations. The high-order bit (12) of the modified 4-bit register designation field is used as an indirect address specifier (I).

Extended Index Addressing

The following four instruction formats are used for index register designation in extended memory operand addressing. (Extended memory operand addressing is described later in this section.)

0 7	8 11	12 14	15
OPC		Rxxx	0

(Extended index register - floating point)

Bits 12-14 of the instruction specify the index register pair Rxxx:RxxxV1 (2-15) in extended floating point indexed register-extended memory operations.

0 7	8 11	12 14	15
OPC		Rx	0

(Extended index register - short instructions)

Bits 12-14 of the instruction specify the index register pair Rx:RxV1 (2-15) in extended short instruction indexed register-memory operations.

0 7	8 11	12	13 14	15
OPC		0	Rx	0

Bits 13-14 of the instruction specify the index register pair Rxx:RxxV1 (2-7) in extended direct/indexed register-memory operations.

0 7	8 11	12	13 15
OPC		I	Rxx

Bits 13-15 of the instruction specify the index register Rxx (1-7) used in extended indirect/indexed register-memory operations.

Multiple-Register Addressing

Double, triple and quadruple-register source or destination operands are addressed by a modified 4-bit register designation field in the instruction. Only the high-order 3 bits are used to address double-registers, and only the high-order 2 bits are used to address triple or quadruple-registers.

The low-order 1 or 2 bits of the register designation field are implied to be zero for multi-register addressing and are coded with explicit control information to augment (A) the operation code (OPC).

0 7	8 10	11	12	15
OPC	Rd	A	Rs	A

(Source and destination double-registers)

0 7	8 9	10 11	12	14 15
OPC	Rd	A	Rs	A

(Source and destination triple-quadruple registers)

Bits 12-14 or 12-13 of the instruction specify the high-order bits of the source double-register or triple/quadruple-register operand, and bits 8-10 or 8-9 specify the high-order bits of the destination double-register or triple/quadruple-register operand in register-to-register operations.

0 7	8	11	12 15
OPC	Rd/Rs	A	

(Source or destination double-registers)

Bits 8-10 of the instruction specify either the source double-register or destination double-register operand in register-memory operations.

Memory Operands/Addressing

Memory Organization

Memory operands may be bit, byte, word, double-word, triple-word, quadruple-word, file, or stack or block operands (see figure 8-4). They are formed in a manner similar to their counterpart register operands except that these operands can start on any word boundary.

Memory stacks consist of a specified number of contiguous memory words starting at the current stack address indicated by associated stack pointers. Memory blocks consist of a specified number of contiguous memory words starting at a specified address in virtual or extended memory.

Memory operands may be accessed through several addressing methods. They may be manipulated with bit constants and with operands in the General Purpose Register File.

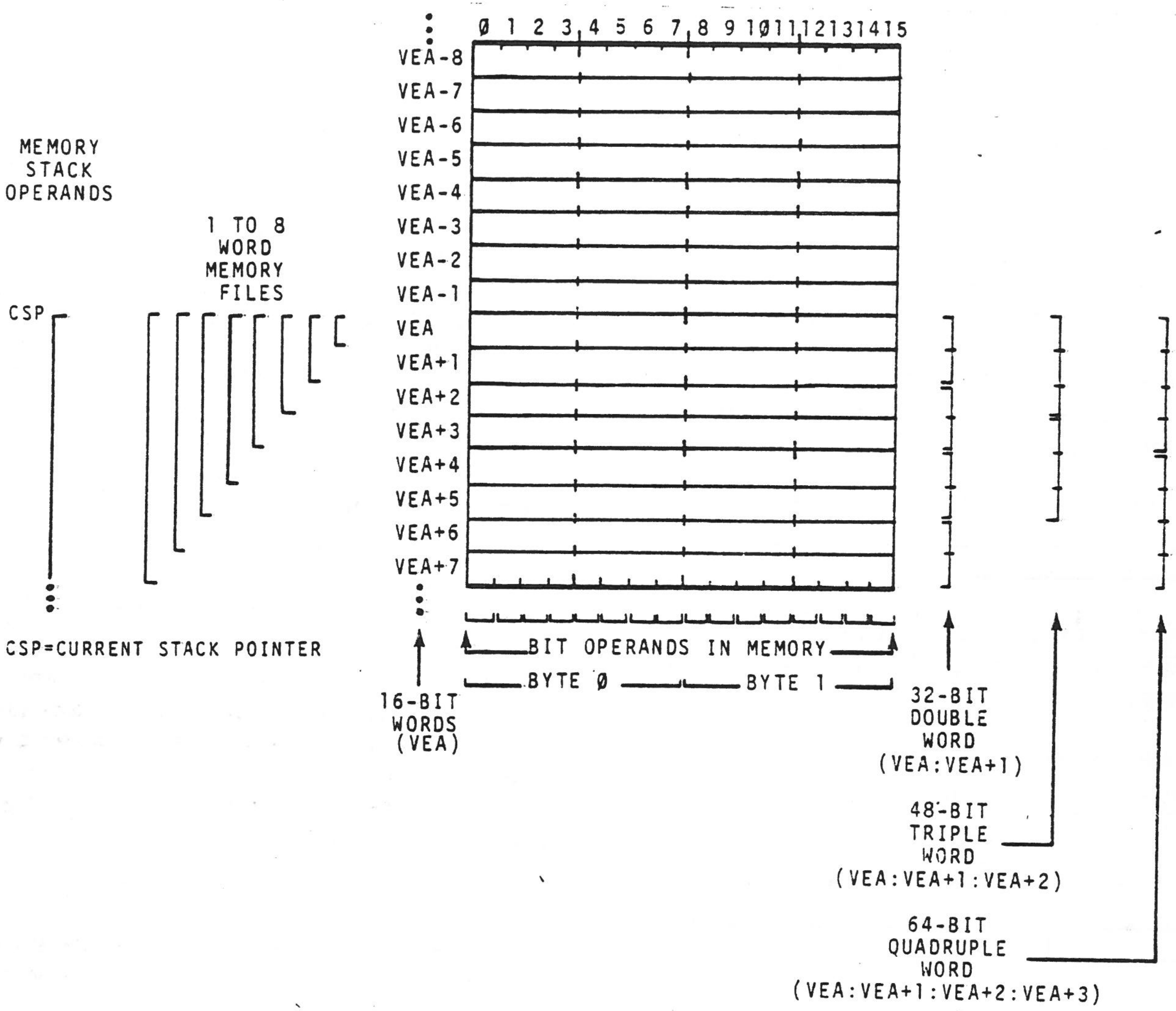
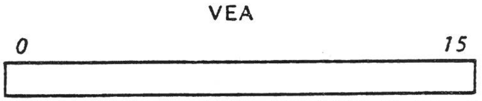


Figure 8-4. Memory Operand Addressing

Single-Word Operands

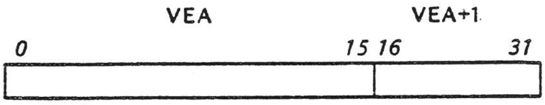
Each memory word may be manipulated separately.



Single-words are designated VEA if virtually addressed, or EEA if actually addressed. A single memory word may contain one 16-bit fixed point, logical or address value, two 8-bit byte values, or sixteen 1-bit binary values. Sets of single words are addressable as memory files or stacks.

Double-Word Operands

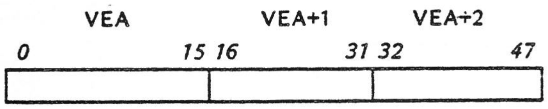
Memory words may be manipulated in concatenated pairs starting at any single-word boundary.



Double-words are designated VEA:VEA+1 or EEA:EEA+1. A double-word may contain one 32-bit fixed point value or one 32-bit floating point value.

Triple-Word Operands

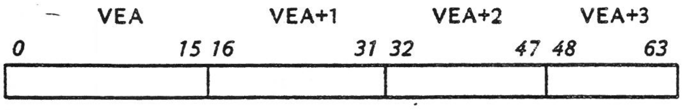
Memory words may be manipulated in triples starting at any single-word boundary.



Triple-words are designated VEA:VEA+1:VEA+2 or EEA:EEA+1:EEA+2. A triple-word may contain one 48-bit floating point value.

Quadruple-Word Operands

Memory words may be manipulated in quadruples starting at any single-word boundary.



Quadruple-words are designated VEA:VEA+1:VEA+2: VEA+3 or EEA:EEA+1:EEA+2:EEA+3. A quadruple-word may contain one 64-bit floating point value, or the extended results of some 32-bit fixed-point operations.

Bit Operands

Individual bits within memory words may be manipulated either arithmetically or logically. Sixteen bits are available in each memory word.

Byte Operands

Individual bytes may be transferred between memory and the general registers. There are two separate 8-bit bytes in each memory word.

Memory File Operands

Up to eight general registers may be transferred between the registers and memory as a file. Any contiguous set of from one to eight memory words may contain a file.

Memory Stack Operands

A memory stack is defined from a low memory address to a high address. Up to 255 associated contiguous memory words may be allocated (added) or deallocated (deleted) at the low end of the stack relative to the Current Stack Pointer (CSP) during a stack operand transfer. From one to sixteen general registers may be transferred between the registers and memory, also relative to the Current Stack Pointer.

Memory Block Operands

When transferring memory between virtually addressed and extended-addressed (extended Addressing) memory, blocks of up to 64K words are defined by a starting address, either virtual or extended and a 16-bit transfer count. This transfer count is incremented as the instruction transfers each 16-bit word and thus defines the size of the block.

Memory Addressing

Virtual addresses are 16 bits long. Extended addresses are 21 bits long. They point to a single-word operand or to the lowest word of a multi-word or file operand.

Virtual operand addresses are relative to the program's virtual operand addressing space, defined by the task's operand map image. Virtual page to actual page conversion is performed during run time by the program's currently selected operand map (OM).

Extended operand addresses are relative to the program's extended operand addressing space, defined by the task's extended memory control word. Extended address to actual address conversion is performed during runtime by adding the computed Extended Effective Address and the contents of the Extended Memory Control Register Lower Boundary.

Virtual branch addresses are relative to the program's virtual instruction addressing space, defined by the tasks instruction map image. Virtual page to actual page conversion during run time is performed by the program's currently selected instruction map (IM).

A memory operand may be a source or a destination operand. Source memory operands are manipulated with source register (Rs) operands. Destination memory operands are manipulated with source register (Rs) operands or with bit constants.

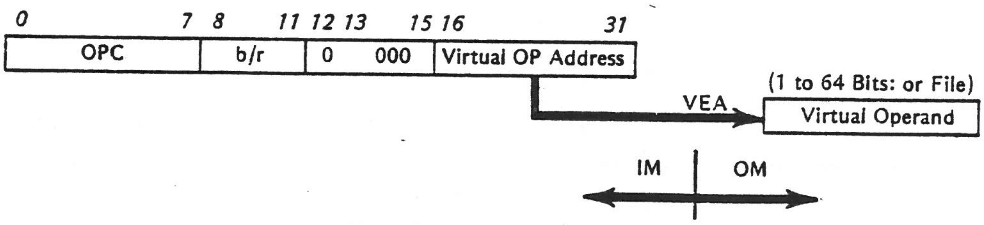
Direct/Indexed/Indirect Addressing

Direct, indexed and/or indirect addressing are the most common addressing modes in the Classic instruction set. These addressing modes apply to bit, word, double-word, triple-word, quadruple-word, and file operands located anywhere in the virtual addressing space of a program's operand map (OM).

Instructions that use these addressing modes consist of two, three or four memory words mapped in the virtual addressing space of the program's instruction map (IM). The second word of these instructions is the direct virtual address of the memory operand. This address may be modified by indexing, by indirect addressing (one level), or by a combinaton of both (indexing first), as specified in the first word of the instruction along with the operation code.

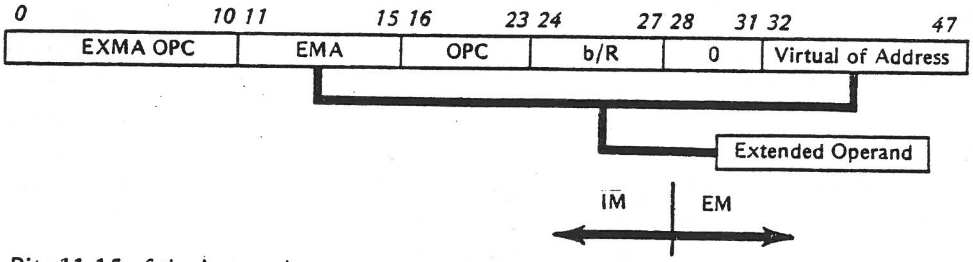
The first instruction word also controls bit selection in memory bit-operations and source or destination register selection for memory-register operations. Indirect operand address words are located in memory relative to the operand virtual addressing space.

DIRECT – The indirect bit (12) is reset to 0 and the index register field (bits 13-15) is zero.



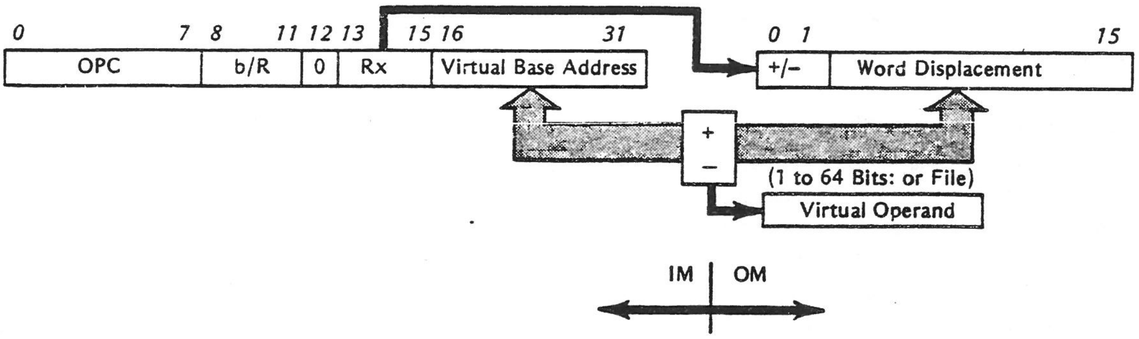
Bits 16-31 of the instruction constitute the VEA and point directly to the virtual operand in the virtual operand addressing space (OM). Bits 8-11 address a bit (b) within the virtual operand in memory bit addressing.

EXTENDED DIRECT – A direct memory reference instruction is immediately preceded by an EXMA instruction.



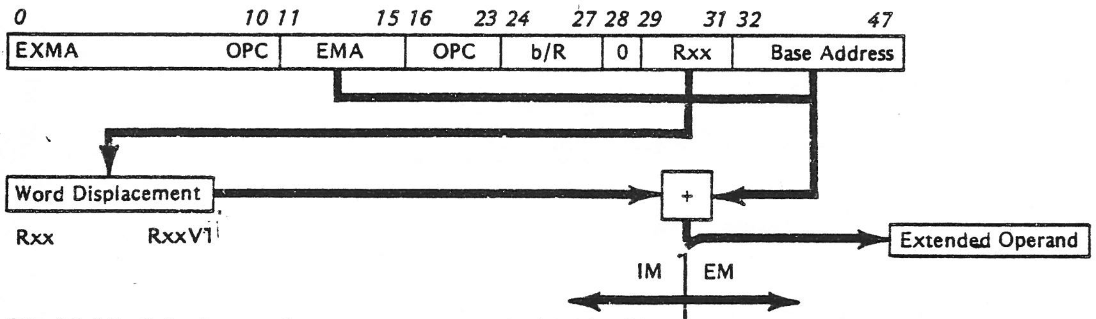
Bits 11-15 of the instruction concatenated with bits 32-47 form the extended effective address, and point directly to a word in extended memory (EM). Bits 24-27 address a bit (b) in the extended operand in memory bit addressing.

INDEXED – The indirect bit (12) is reset to 0 and the index register field (bits 13-15) specifies a general register (Rxx) in the range 1-7. The address field (bits 16-31) is a virtual base address. The specified index register contains a signed word displacement value in the range –32,768 to +32,767.



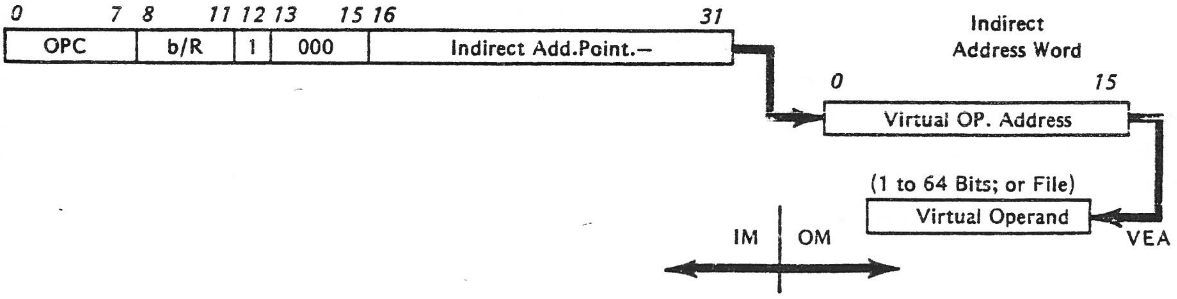
Bits 16-31 of the instruction are added to the contents of the specified index register to form the VEA of the virtual operand in the virtual operand addressing space (OM). Bits 8-11 address a bit (b) within the virtual operand in memory bit addressing.

EXTENDED INDEXED – An indexed memory reference is immediately preceeded by an EXMA instruction.



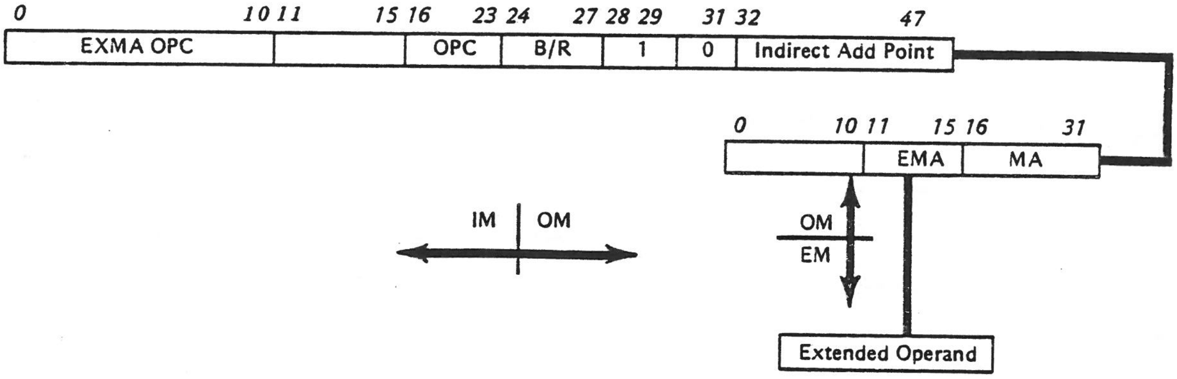
Bits 11-15 of the instruction are concatenated with bits 32-47 of the instruction to form the extended base address. This is added to the two's complement word displacement in the concatenated index register pair Rxx:RxxV1 to form the extended effective address, which points directly to a word in extended memory space (EM). Bits 24-27 address a bit (b) in the extended operand in memory bit addressing.

INDIRECT – The indirect bit (12) is set to 1 and the index register field (bits 13-15) is zero.



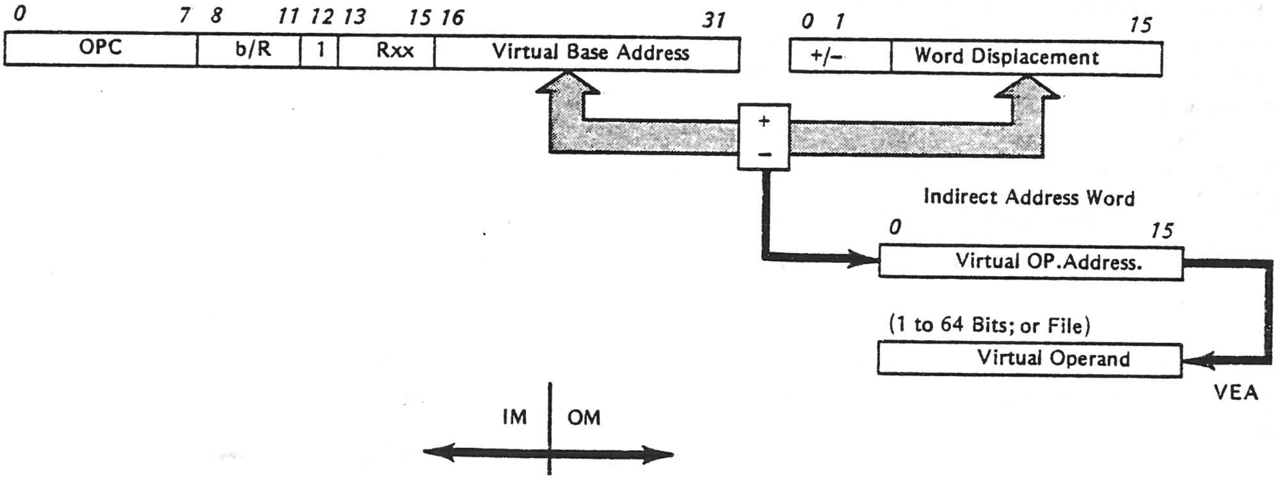
Bits 16-31 of the instruction point to an indirect address word in the virtual operand addressing space (OM), which in turn contains the VEA which points to the virtual operand in the same addressing space. Bits 8-11 address a bit (b) within the virtual operand in memory bit addressing.

EXTENDED INDIRECT – An indirect memory reference instruction is immediately preceeded by an EXMA instruction.



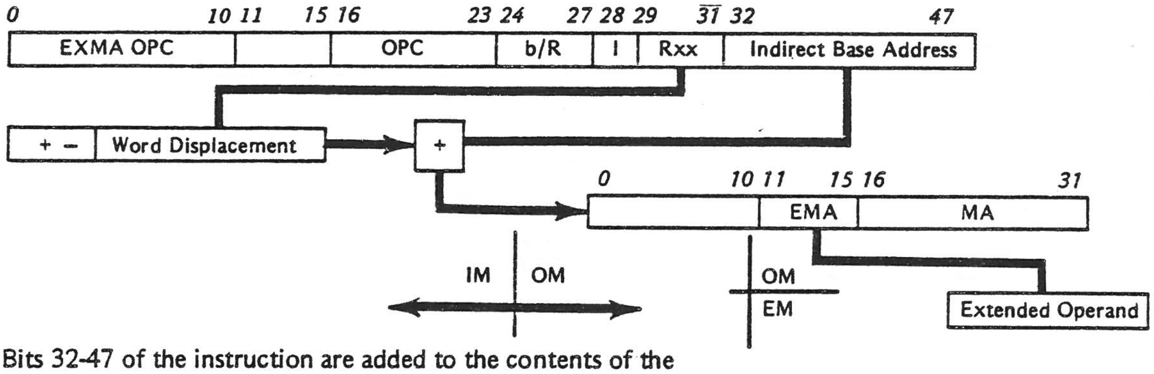
Bits 32-47 of the instruction point to a doubleword in the virtual operand space (OM), which points to a word in extended memory space (EM). Bits 24-27 address a bit (b) in the extended operand in memory bit addressing.

INDEXED/INDIRECT – The index bit (12) is set to 1 and the register field (bits 13-15) specifies a general register (Rxx) in the range 1-7.



Bits 16-31 of the instruction are added to the contents of the specified index register to form a pointer to an indirect address word in the virtual operand addressing space (OM), which in turn contains the VEA that points to the virtual operand in the same addressing space. Bits 8-11 address a bit (b) within the virtual operand in memory bit addressing.

EXTENDED INDEXED/INDIRECT – An indexed/indirect memory reference is immediately preceded by an EXMA instruction.

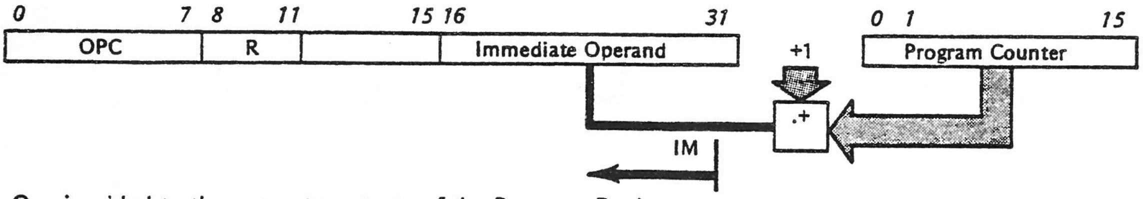


Bits 32-47 of the instruction are added to the contents of the specified index register to form a pointer to an indirect address doubleword in the virtual addressing space, which in turn contains the 21-bit address of the operand in extended memory space. Bits 24-27 address a bit (b) in the extended operand in memory bit addressing.

Immediate Addressing

A word operand that is contained within a referencing instruction may be addressed implicitly.

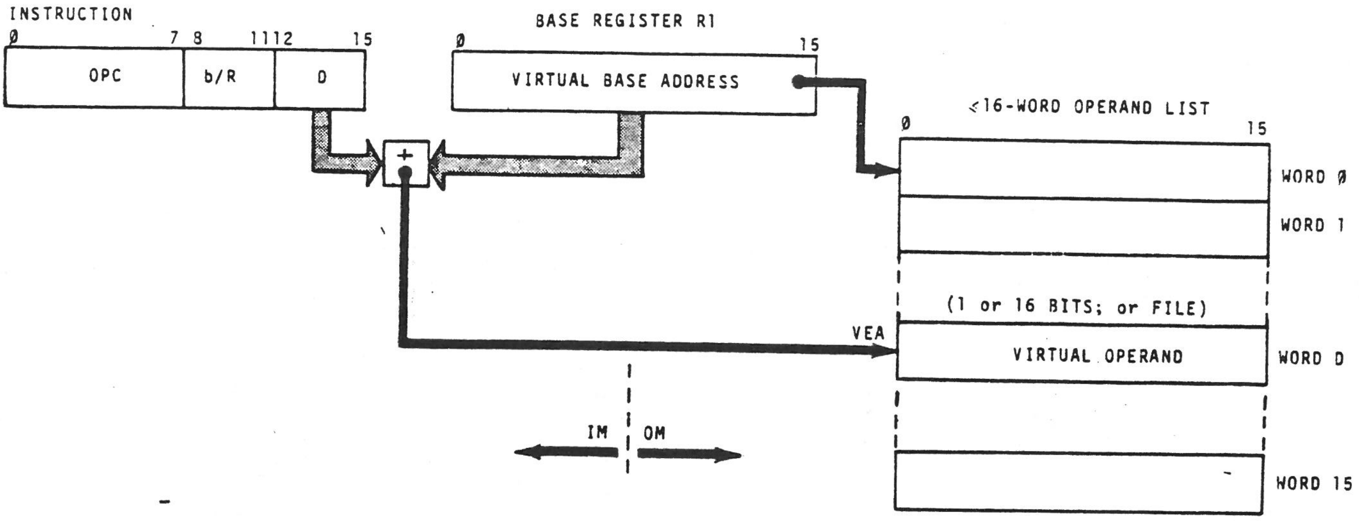
Instructions that use this addressing mode consist of two memory words mapped in the virtual addressing space of the program's instruction map (IM). The second word of the instruction contains the immediate operand. These are the only operands that are accessed relative to the program's virtual instruction addressing space (IM).



One is added to the current contents of the Program Register to form the VEA of the immediate operand in bits 16-31 of the current instruction in the virtual instruction addressing space (IM).

Short-Displaced Addressing

Bit, word or file operands contained in a maximum 16-word table located anywhere in the virtual addressing space of the program's operand map (OM) may be accessed relative to its base (beginning) address contained in register R1.



Instructions that use this memory addressing mode consist of of one, two, or three memory words mapped in the virtual addressing space of the program's instruction map (IM). In addition to the operation code, the first instruction word specifies the displacement (D) of the memory operand from the beginning of the memory table designated by the virtual base address contained in register R1. It also controls bit selection in memory bit operations, and source or destination register selection for memory-register operations.

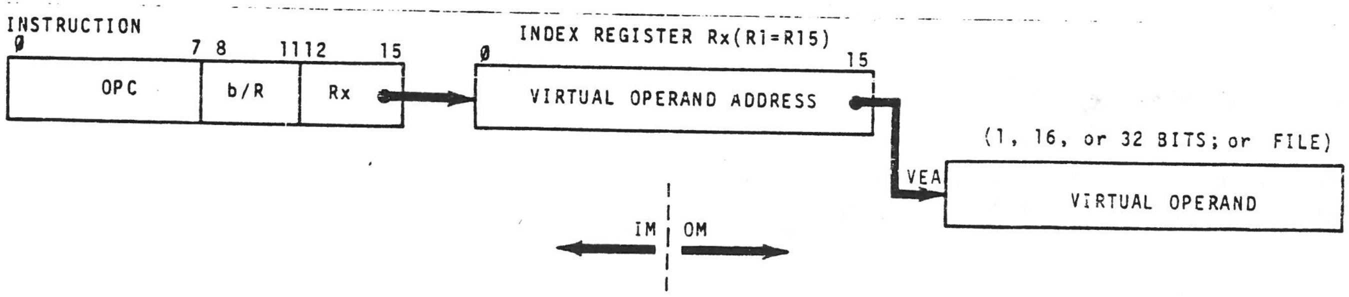
Bits 12-15 of the instruction are added to the contents of base register R1 to form the VEA of the virtual operand in the virtual operand addressing space (OM). Bits 8-11 address a bit (b) within the virtual operand in memory bit addressing.

Short-Indexed Addressing

Bit, word, double-word, and file operands located anywhere in virtual or actual addressing space may be accessed via the address contained in any of the 15 general registers.

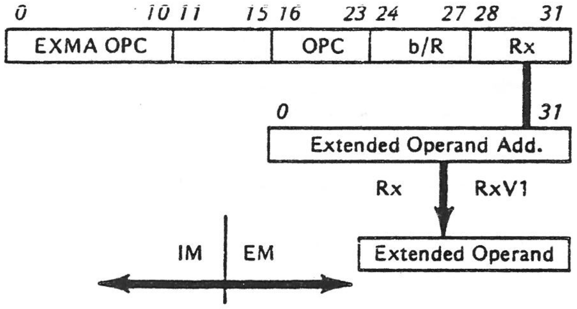
Instructions that use this memory addressing mode consist of one, two, or three memory words mapped in the virtual addressing space of the program's instruction map (IM). In addition to the operation code, the first instruction word specifies the general register(s) that contains the effective virtual address of the memory operand. It also controls bit selection in memory bit operations, and source or destination register selection for memory-register operations.

Nonvirtual and Virtual Mode:



Bits 12-15 of the instruction specify a general register (Rx) that contains the VEA of the virtual operand in the virtual operand addressing space (OM). Bits 8-11 address a bit (b) within the virtual operand in memory bit addressing.

EXTENDED SHORT INDEXED ADDRESSING – A short indexed memory reference is immediately preceded by an EXMA instruction.



Bits 28-31 specify a register pair which contains the 21-bit address of an operand in extended memory space (EM). Bits 24-27 address a bit (b) in the extended operand in memory bit addressing.

Bit Addressing

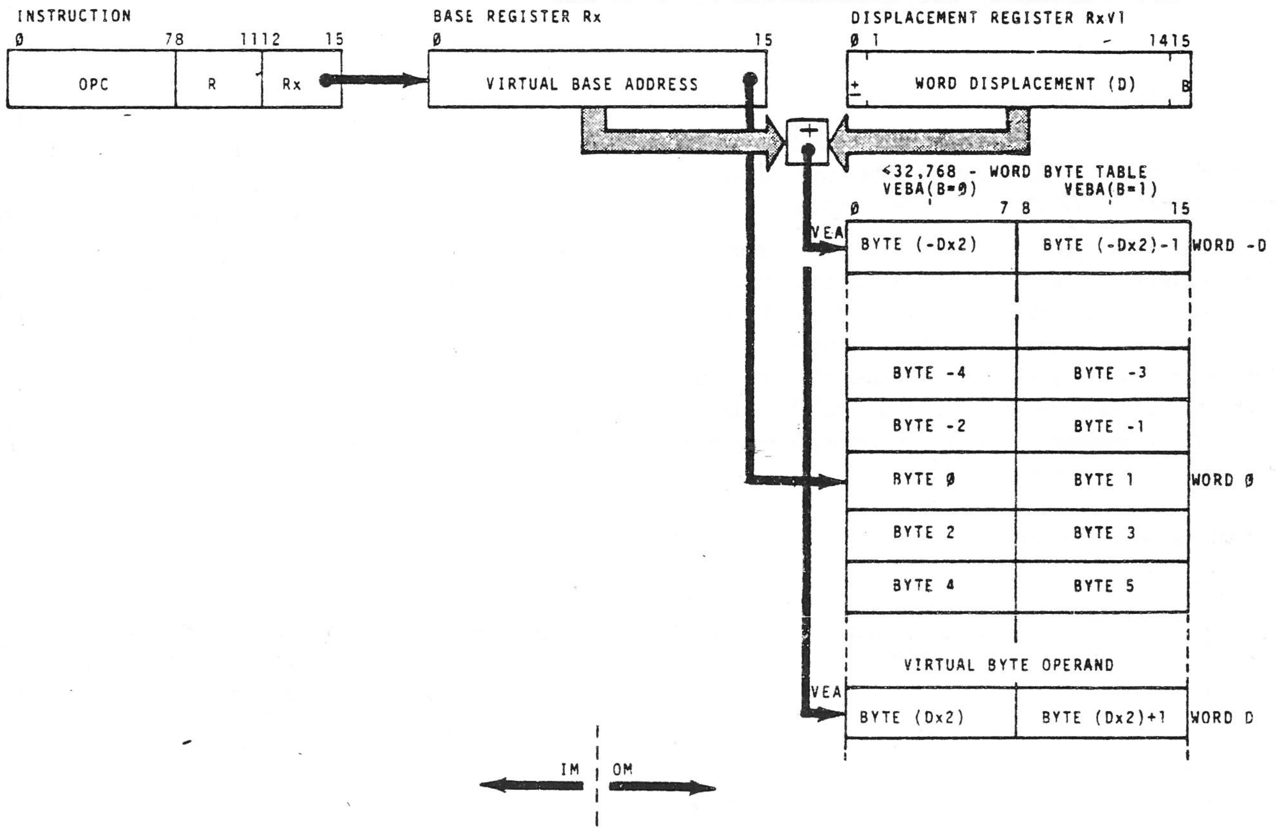
Addressing of individual bits within virtual operands in memory is provided (and described) in each of the preceding memory operand addressing modes, except immediate operand addressing. Bit addressing applies to destination operands only, and is performed with logical bit constants of bit value "1" or with 16-bit values of "powers-of-2." In all cases, the bit designator (b) is a 4-bit field in bit positions 8-11 of the first word of the instruction.

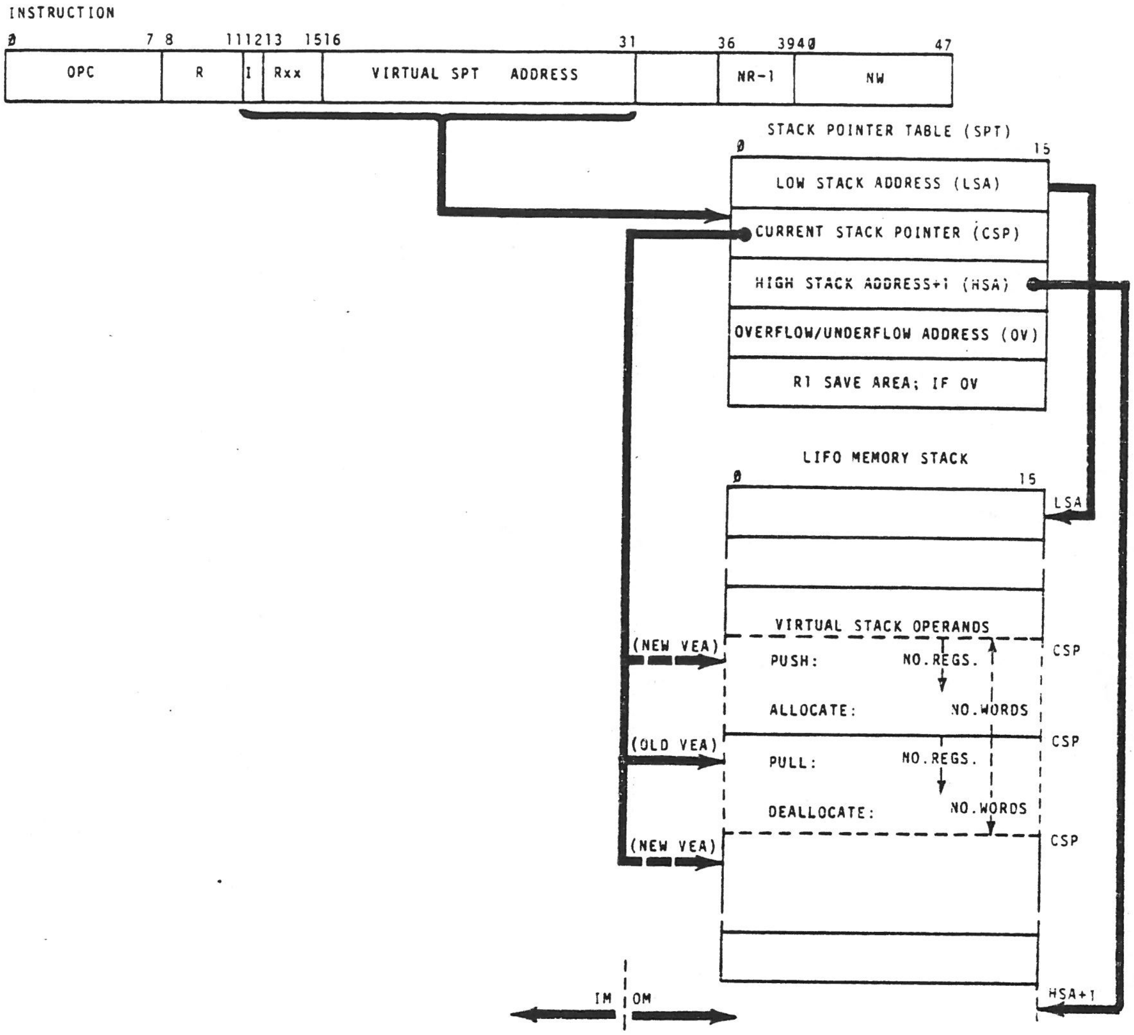
Byte Addressing

Byte operands contained in a contiguous byte table located anywhere in the virtual addressing space of a program's operand map (OM) may be accessed relative to its base (middle) address as specified in a pair of general registers.

Instructions that use this memory addressing mode consist of one memory word mapped in the virtual addressing space of the program's instruction map (IM). In addition to the operation code and source or destination register selection, the instruction word specifies the register double-word (Rx:RxV1) that contains the effective virtual base address of the middle of the byte-string, the plus or-minus displacement of the memory-word containing the desired byte, and the byte designator (B).

Bits 12-15 of the instruction specify the byte base/displacement addressing register pair. The sign-extended contents of bits 0-14 of displacement register RxV1 are added to the contents of base register Rx to form the VEA of the virtual operand word in the virtual operand addressing space (OM). Bit 15 of displacement register RxV1 addresses either the upper byte (B=0) or the lower byte (B=1) which becomes the effective byte virtual operand (VEBA).





Stack Addressing

Stack operands (to be) contained in a contiguous last-in-first-out out (LIFO) push/pull stack located anywhere in the virtual addressing space of a program's operand map (OM) may be stored or accessed in allocated or deallocated stack space relative to the current position in the stack as specified in a separate stack pointer table (SPT) in memory.

Instructions that use this memory addressing mode consist of three memory words mapped in the virtual addressing space of the program's instruction map (IM). In addition to the operation code and the source or destination register selection, the first two instruction words specify the location of the SPT plus 1 in memory using any of the first four standard addressing modes (direct/indexed/indirect) previously described. The third instruction word designates the number of words in the memory stack to allocate or deallocate (from 0 to 255) and the number of general registers minus 1 to transfer from 1 to 16.

Bits 12-31 of the instruction determine the effective virtual address of the stack pointer table plus 1 (SPT+1) in the virtual operand addressing space (OM). Words one and three of the SPT point to the beginning (LSA) and end (HSA) of the memory stack. Word two points to the current top of the stack (CSP), which always is the VEA of its virtual operands within the virtual operand addressing space (OM). Bits 40-47 of the third instruction word specify the number of words (NW) to allocate when "pushing" or to deallocate when "pulling" the number of register (NR) specified (minus 1) in bits 36-39 of the instruction.

NR-1 must be smaller than NW. NW is subtracted from CSP (the old VEA) to form the new VEA prior to a push operation. Following a pull operation NW is added to CSP to form the new VEA. The new VEA must not become smaller than LSA (overflow) or larger than HSA (underflow).

Other Addressing

The preceding memory operand addressing modes are used by the majority of the Classic instructions. Descriptions of these instructions follow. Six additional memory operand addressing modes are available in the Classic. They are described later in this chapter with the instructions that invoke them:

Hop Instructions

- Relative displaced addressing

Control Instructions (privileged)

- Map Image addressing
- Map and Map Image addressing
- Map addressing
- Via-Map-Image addressing
- Actual Memory addressing

All other operand addressing in the Classic is used to perform specific (and privileged) processor or memory control functions. These are also described in the following section with their appropriate instructions.

Extended Memory Addressing Instructions

Most indexed and short-indexed instructions are executable in the Extended Memory addressing mode. For quick reference, these instructions are tabulated below:

ABMM, ABXM, ABMB, ABXB
ADM, ADMM, ADX, ADXM, ADMB, ADXB, ADMD
CBMB, CBXB
CLM, CLMD
CRM, CRX, CRMB, CRXB, CRMD, CRMT, CRMQ
DVM, DVX, DVMD
ETMM, ETXM, ETM, ETX, ETMB, ETXB
FAM, FAMD, FAMQ
FDM, FDMD, FDMQ
FMM, FMMD, FMMQ
FSM, FSMD, FSMQ
IRM
LDM, LDX, LDMD, LDXD, STM, STX, STMD, STXD
LDMT, LDMQ, STMT, STMQ
LFM, SFM, LFX, SFX
MBEV, MBVE
MPM, MPX, MPMD
OBMM, OBXM
ORMM, ORXM, ORM, ORX
SUM, SUX, SUMD
TBMM, TBXM, TBMB, TBXB
TRMB, TRXB
TSBM
XOM, XOX
ZBMM, ZBXM, ZBMB, ZBXB

See page 9-125 for EXMA instructions.

Chapter 9. INSTRUCTION SET

All CLASSIC instructions are described in this section. The instructions are grouped in ten functional classes:

- Move
- Fixed Point Arithmetic
- Floating Point.Arithmetic
- Logical
- Shift
- Compare and Test
- Branch and Hop
- Control
- Interrupt and Call
- Input/Output

Appendix D, Part 1, is an alphabetical listing of instructions in order of the instruction mnemonic. Information pertaining to page location, execution time and operation code is also contained in Part 1. Part 2 is a cross reference chart from operation code to mnemonic.

Symbols and Abbreviations

The symbols and abbreviations used in the instruction descriptions are listed below in alphabetical order. Where a second symbol in parenthesis is shown following a symbol, it is the Assembler Language equivalent designation for the preceding hardware symbol.

A — Virtual operand address

AA — Actual Address, which is a 21-bit physical address that results from a 16-bit VEA going through the mapping logic

ACT — Interrupt Active

AEA — Actual Effective Address, applicable to actual address in Extended Addressing Mode.

B — Virtual Branch address

b — Bit position (0-15)

CCC — Condition Code designating adder carry out in result of instruction execution

CCN — Condition Code designating negative rerult of instruction execution

CCO — Condition Code designating overflow result of instruction execution

CCZ — Condition Code designating zero result of instruction execution

c — Shift count (0-15)

D (d) — Displacement value (0-15), which is used in the short-displaced addressing mode

EEA — Extended Effective address, virtually mapped operand address in extended memory (prior to adding EMCR Lower boundary).

ENA — Interrupt Enable

FLT — Floating point operand

G — I/O group number

h — Hop displacement (-64 to +63)

I (*) — Indirect address bit

INT — Integer operand

IR — Instruction Register

L — Length of (number of words in) Map and/ or Map Image to process

l — Interrupt level (0-15)

NR (N) — Number of registers transferred

NW (W) — Number of words allocated to or deallocated from a memory stack

NB (B) — Number of words in a block transferred between Virtual and Actual memory

NZOC — All four Condition Codes as a single hex value denoting either standard composite conditions of special character classifications of bytes in registers

EMCR — Extended Memory Control Register, holds upper and lower modulo-8K boundaries used during extended memory address calculation

n — Execute service number (0-255)

PR — Program Register, which is a 16-bit register containing the current program location

PS — Program Status which is the first word of the PSD

PSD — Program Status Doubleword

QNT — Quotient in result of division operation

R — A general register which is not explicitly a destination operand or source operand

R:RV1 — A register doubleword containing the concatenated values stored in register R (most significant half) and register R OR'ed with one (RV1) (least significant half), where R is a even numbered register not zero.

R:RV1:RV2 — A register triple-word containing the concatenated values stored in register R (most significant), where R is register 4, 8 or 12

R:RV1:RV2:RV3 — A register quadruple-word containing the concatenated values stored in register R (most significant), R OR'ed with one, R OR'ed with two, and R OR'ed with three (least significant), where R is register 4, 8 or 12

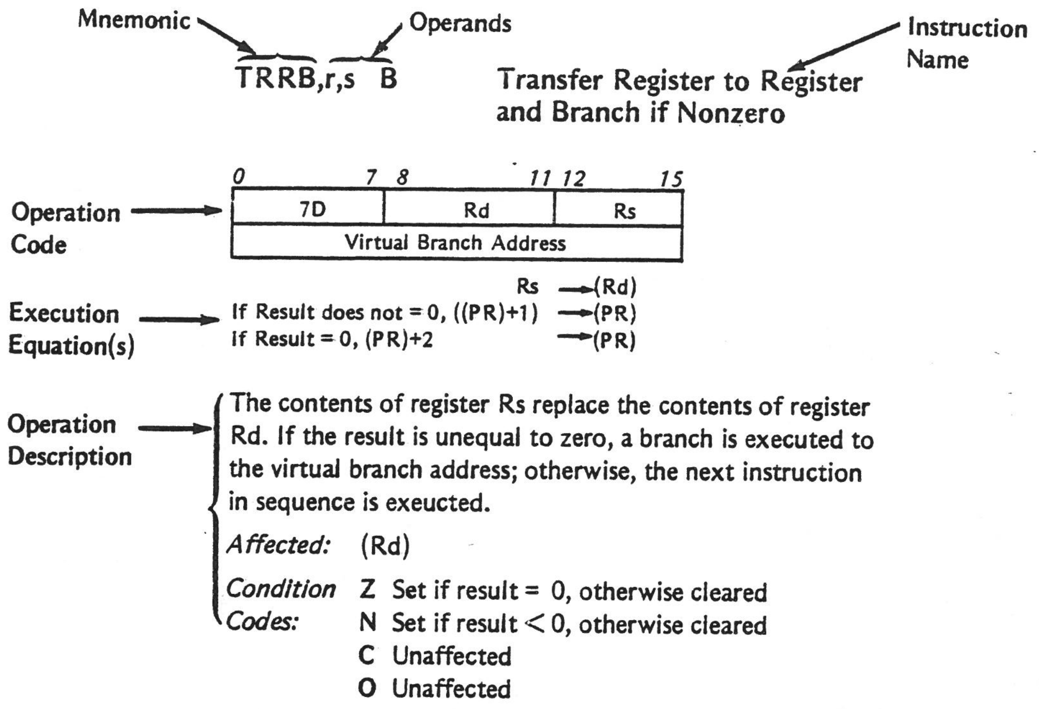
RB	– Current active working Register Block in Register Context File
RD	– Relative Displacement value in Hop instructions
Rd(r)	– General register Rd, which is the operand destination register for many instructions
Rs(s)	– General register Rs, which is the operand source register for many instructions
Rx(x)	– Effective address register for short indexed instructions (1⩽x⩽15)
Rxx(x)	General register Rxx (1⩽xx⩽7), which is the index register for many instructions. When Rxx = 0, no index operation occurs
Rxxx(x)	– Same as Rxx except many be extended to (1⩽xxx⩽15) for floating-point instuctions
R1	– Dedicated base register R1
S	– Sign bit
SPT	– Stack Pointer Table in memory containing low, high and current addresses in a memory stack, along with an overflow/underflow exit address and an R1 save area
U	– I/O unit number (0-15)
us	– Microseconds
V	– Immediate operand value
VEA	– Virtual effective memory address of word operand, which is the address that results after all specified address manipulation operations have been completed except mapping
VEA:VEA+1	– Virtual Effective Memory Address of doubleword operand
VEA:VEA+1: VEA+2	– Virtual Effective Memory Adress of Tripleword operand
VEA:VEA+1: VEA+2: VEA+3	– Virtual Effective Memory Address of Quadrupleword operand
VEBA	– Virtual effective byte memory address, which is the upper or lower byte of the VEA
(CC)Z	– Condition Code designating zero result of instruction execution
( )	– Contents of
→	– Replace the contents of
+	– Addition operator
—	– Subtraction operator
x	– Multiplication operator
/	– Division operator
:	– Concatenation operator
.AND.	– Logical AND operator
v	– Logical OR operator
.XOR.	– Logical Exclusive OR operator
.LE.	– Less than or equal to
.GE.	– Greater than or equal to
.GT.	– Greater than
.LS.	– Less than
$\overline{\text{operand}}$	– Logical NOT (One's complement) operator

Instruction Format

Instructions are organized and described by major functional class.

Instructions that perform identical sub-functions within a class and simply manipulate different operand type or use different memory operand addressing modes are described together within sub-class groups. Each group is introduced by a general functional description.

Each instruction within a group is then described in detail. The general format for an individual instruction description is:



TRRB,r,s B

Transfer Register to Register and Branch if Nonzero

0 — 7	8 — 11	12 — 15
7D	Rd	Rs
Virtual Branch Address		

Rs → (Rd)
If Result does not = 0, ((PR)+1) → (PR)
If Result = 0, (PR)+2 → (PR)

The contents of register Rs replace the contents of register Rd. If the result is unequal to zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is exeucted.

Affected: (Rd)

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set if result < 0, otherwise cleared
C Unaffected
O Unaffected

- The **Mnemonic** is a three or four letter representation of the instruction name. (The asterisk, when present, indicates indirectability.)
- The **Operands** of the instruction are expressed in Assembler Language notation and format.
- The **Instruction Name** briefly describes the function performed by the execution of the instruction, including the memory addressing mode when applicable and the operand type.
- The **Operation Code** value is shown as two hexadecimal digits in bit positions 0-7 of the first instruction word. It controls the instruction execution function, memory addressing mode, and operand type.
- Bits 8-15 of the first instruction word contain binary coded register addresses in many instructions and other binary coded fields in other instructions, as described.
- Many instructions contain a second, some a third, and few a fourth instruction word used for an immediate operand, 16-bit Virtual Effective Memory Address(es) or special control fields. The address in the Program Register (PR) referenced in the description of these instructions is that of the first instruction word.
- The **Execution Equation(s)** is a logical expression of all program controlled functions performed in the computer which comprise the instruction execution.
- Except for stack, branch and hop instructions, the contents of the Program Register always are advanced to the first word of the next instruction. Special modifications to the Program Register are noted where applicable.
- The **Operation Description** paragraph describes all program controlled functions performed in the computer which comprise the instruction execution. In addition, the contents of the Program Register are advanced to the first word of the next instruction.
- The *Affected* line lists all general registers, memory cells and programmable control registers in which the contents are modified as a result of the execution of the instruction. Source operands are not modified. The contents of the Program Register always are affected and therefore not mentioned. Modifications to Condition Codes are described separately.
- The *Condition Codes* line lists the status of the four conditions that can result from the instruction execution: Zero (Z), Negative (N), Carry (C), Overflow (O). If the instruction does not affect Condition Codes, this is noted. Most instructions modify the Condition Codes.

Move Instructions

This instruction class provides the capability of moving bit constants to the general purpose registers and of moving byte, word and multi-word operands from memory to the registers, from the registers to memory and from register(s) to register(s).

A bit, byte, word, doubleword, quadruple-word, file of one to eight words, or stack entry of one to sixteen words can be moved by a single instruction execution. The memory operations include most addressing modes. Optional unconditional branching or branching on nonzero result is provided with some of these instructions. Most of the instructions in this class set Condition Codes.

Load Bit(s) in Register (and Branch Unconditionally)

The following sub-group of Move instructions is used toload a single bit in a register (power-of-two) or generate a bit string mask in a register starting from bit 0 (negative power-of-two). The remaining bits in the register are cleared to zero. Either of these operations may be specified to branch unconditionally.

LBR,r,b — Load Bit in Register

0 7	*8 11*	*12 15*
65	Rd	b

$1 \rightarrow Rd_b$
$0's \rightarrow Rd_{0-(b-1)}$
$0's \rightarrow Rd_{(b+1)-15}$

A one is stored in register Rd in bit position b. Zeros are stored in all other bit positions in register Rd.

Affected: (Rd)

Condition Codes:
Z Cleared
N Set if bit 0 of result = 1, otherwise cleared
C Unaffected
O Unaffected

LBRB,r,b B — Load Bit in Register and Branch Unconditionally

0 7	*8 11*	*12 15*
75	Rd	b
Virtual Branch Address		

$1 \rightarrow Rd$
$0's \rightarrow Rd_{0-(b-1)}$
$0's \rightarrow Rd_{(b+1)-15}$
$((PR) + 1) \rightarrow (PR)$

A one is stored in register Rd in bit position b. Zeroes are stored in all other bit positions in register Rd. A branch is then executed unconditionally to the virtual branch address.

Affected: (Rd)

Condition Codes:
Z Cleared
N Set if bit 0 of result = 1, otherwise cleared
C Unaffected
O Unaffected

Appropriate Condition Codes are set if bit 0 of the result is set (negative) or if the generate mask operation specifies bit 0 (overflow).

GMR,r,b — Generate Mask in Register

0 7	*8 11*	*12 15*
67	Rd	b

$1's \rightarrow (Rd)_{0-b}$
$0's \rightarrow (Rd)_{(b+1)-15}$

Ones are stored in register Rd in bit position Rd_0 through Rd_b. Zeros are stored in register Rd in bit positions Rd_{b+1} through Rd_{15}.

Affected: (Rd)

Condition Codes:
Z Cleared
N Set if bit 0 of result = 1, otherwise cleared
C Unaffected
O Set if the b field (bits 12-15 of the instruction word) is zero, otherwise cleared

GMRB,r,b B — Generate Mask in Register and Branch Unconditionally

0 7	*8 11*	*12 15*
77	Rd	b
Virtual Branch Address		

$1's \rightarrow (Rd)_{0-b}$
$0's \rightarrow (Rd)_{(b+1)-15}$
$((PR) + 1) \rightarrow (PR)$

Ones are stored in register Rd in bit positions Rd_0 through Rd_b. Zeroes are stored in register Rd in bit positions Rd_{b+1} through Rd_{15}. A branch is then executed unconditionally to the virtual branch address.

Affected: (Rd)

Condition Codes:
Z Cleared
N Set if bit 0 of result = 1, otherwise cleared
C Unaffected
O Set if the b field (bits 12-15 of the instruction word) is zero, otherwise cleared

Move Byte Register to Register

The following instructions are used to move a byte within a register or between registers. An upper byte may be shifted right, a lower byte may be shifted left, or the byte may retain its original relative position during the move.

The non-affected byte in the destination register is cleared to zero. If the same register is specified for both source and destination, the contents of the original register are changed, and the result is equivalent to an 8-bit logical shift or a byte clear operation.

Appropriate Condition Code values are set if the result is zero or negative for byte transfers to the upper byte position. Special Condition Code values are set to classify bytes transferred to the lower byte position whose 8-bit codes are ASCII special characters.

MBR,r,s — Move Byte Right Register to Register

0 7	8 11	12 15
08	Rd	Rs

$(Rs)_{0-7} \rightarrow (Rd)_{8-15}$
$0's \rightarrow (Rd)_{0-7}$

The more significant byte conatined in register Rs is transferred to the less significant byte position of register Rd. Zeroes are transferred to the more significant byte position in register Rd.

Affected: (Rd)

Condition Codes:

NZOC		NZOC	
1000	If byte in bits 8-15 is (	1110	If byte is .
1001	If byte is)	1111	If byte is /
1010	If byte is *	0000	If byte is alphabetic
1011	If byte is +	0001	Not classified
1100	If byte is ,	0011	If byte is space
1101	If byte is –	0101	If byte is numeric

MBL,r,s — Move Byte Left Register to Register

0 7	8 11	12 15
09	Rd	Rs

$(Rs)_{8-15} \rightarrow (Rd)_{0-7}$
$0's \rightarrow (Rd)_{8-15}$

The less significant byte contained in register Rs is transferred to the more significant byte position of register Rd. Zeroes are transferred to the less significant byte position in register Rd.

Affected: (Rd)

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set if bit 0 of result = 1, otherwise cleared
C Cleared
O Cleared

MUR,r,s — Move Upper Byte Register to Register

0 7	8 11	12 15
08	Rd	Rs

$(Rs)_{0-7} \rightarrow (Rd)_{0-7}$
$0's \rightarrow (Rd)_{8-15}$

The more significant byte contained in register Rs is transferred to the more significant byte position of register Rd. Zeroes are transferred to the less significant byte position in register Rd.

Affected: (Rd)

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set if bit 0 of result = 1, otherwise cleared
C Cleared
O Cleared

MLR,r,s — Move Lower Byte Register to Register

0 7	8 11	12 15
0C	Rd	Rs

$(Rs)_{8-15} \rightarrow (Rd)_{8-15}$
$0's \rightarrow (Rd)_{0-7}$

The less significant byte contained in register Rs is transferred to the less significant byte position of register Rd. Zeroes are transferred to the more significant byte position in register Rd.

Affected:

Condition Codes:

NZOC					
1000	If byte in bits 8-15 is (	1100	If byte is ,	0001	Not classified
		1101	If byte is –	0011	If byte is space
1001	If byte is)	1110	If byte is .	0101	If byte is numeric
1010	If byte is *	1111	If byte is /		
1011	If byte is +	0000	If byte is alphabetic		

Transfer Register(s) to Register(s) and Branch if Nonzero

The following instructions are used to transfer word, double-word or quadruple-word operands between registers. If the same register is specified for both source and destination a non-operation will occur. Word transfers may be specified to branch if the operand is not zero.

Appropriate Condition Codes are set if the operand is zero or negative.

TRR,r,s — Transfer Register to Register

0 7	8 11	12 15
6D	Rd	Rs

(Rs) → (Rd)

The contents of register Rs replace the contents of register Rd.

Affected: (Rd)

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set if result <0, otherwise cleared
C Unaffected
O Unaffected

TRRB,r,s B — Transfer Register to Register and Branch if Nonzero

0 7	8 11	12 15
7D	Rd	Rs
Virtual Branch Address		

Rs → (Rd)
If Result does not = 0, ((PR)+1) → (PR)
If Result = 0, (PR)+2 → (PR)

The contents of register Rs replace the contents of register Rd. If the result is unequal to zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: (Rd)

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set if result <0, otherwise cleared
C Unaffected
O Unaffected

TRRD,r,s — Transfer Double Register to Double Register

0 7	8	11	12	15
CD	Rd	0	Rs	0

(Rs:RsV1) → (Rd:RdV1)

The contents of registers Rs:RsV1 replace the contents of registers Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:
Z Set if all registers loaded = 0, otherwise cleared
N Set if bit 0 of Rd = 1, otherwise cleared
C Unaffected
O Unaffected

TRRQ,r,s — Transfer Quadruple-Register to Quadruple-Register

0 7	8	11	12		15
8D	Rd	0 0	Rs	0	0

(Rs:RsV1:RsV2:RsV3) → (Rd:RdV1:RdV2:RdV3)

The contents of registers Rs:RsV1:RsV2:RsV3 replace the contents of registers Rd:RdV1:RdV2:RdV3.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:
Z Set if all registers loaded = 0, otherwise cleared
N Set if bit 0 of Rd = 1, otherwise cleared
C Unaffected
O Unaffected

Interchange Bytes Register to Register

The following instruction is used to interchange the positions of both bytes within a register or between registers. If the same register is specified for both source and destination, the result is equivalent to an 8-bit rotate operation. If different registers are specified, the contents of the original register are preserved and the rotated bytes occupy the new register.

Special Condition Code values are set to classify the byte transferred to the lower byte position if its 8-bit code is an ASCII special character.

IBR,r,s — Interchange Bytes Register to Register

0 – 7	*8 – 11*	*12 – 15*
0A	Rd	Rs

$(Rs)_{0-7} \rightarrow (Rd)_{8-15}$

$(Rs)_{8-15} \rightarrow (Rd)_{0-7}$

The less significant byte contained in register Rs is transferred to the more significant byte position in register Rd, and the more significant byte contained in register Rs is transferred to the less significant byte position in register Rd.

Affected: (Rd)

Condition Codes:	NZOC	NZOC
	1000 If byte is (	1110 If byte is .
	1001 If byte is)	1111 If byte is /
	1010 If byte is *	0000 If byte is alphabetic
	1011 If byte is +	0001 Not classified
	1100 If byte is ,	0011 If byte is space
	1101 If byte is -	0101 If byte is numeric

Interchange Register and Operand

The following instructions are used to interchange a word or double-word in a register with another word or double-word either in a register or in memory. If source and destination addresses differ, the source word(s) is transferred to the destination register and the destination word(s) is transferred to the source register or source memory location(s). If the same register(s) is specified for both source and destination, a nonoperation will occur.

Appropriate Condition Codes are set if the final destination word is zero or negative.

IRR,r,s — Interchange Register and Register

0 7	8 11	12 15
B7	Rd	Rs

(Rs) → (Rd)
(Rd) → (Rs)

The contents of register Rd are interchanged with the contents of register Rs.

Affected: (Rd), (Rs)

Condition Codes:
Z Set if (Rd) = 0, otherwise cleared
N Set if (Rd) < 0, otherwise cleared
C Unaffected
O Unaffected

IRM*,r,x A — Interchange Register and Memory

0 7	8 11	12	15
B6	Rd	I	Rxx
Virtual Operand Address			

(VEA) → (Rd)
(Rd) → (VEA)

The contents of memory location VEA are interchanged with the contents of register Rd.

During execution of this instruction the CPU inhibits any other memory user (excluding Input/Output traffic) from initiating a memory call. This prevents the contents of memory location VEA from being altered by another memory user during execution of the IRM

Affected: (Rd), (VEA)

Condition Codes:
Z Set if (Rd) = 0, otherwise cleared
N Set if (Rd) < 0, otherwise cleared
C Unaffected
O Unaffected

IRRD,r,s — Interchange Double Register and Double Register

0 7	8 10	11	12	15
8A	Rd	1	Rs	0

(Rs:Rsv1) → (Rd:Rdv1)
(Rd:Rdv1) → (Rs:Rsv1)

The contents of register pair Rd:Rdv1 are interchanged with the contents of register pair Rs:Rsv1.

Affected: (Rd:Rdv1),(Rs:Rsv1)

Condition Codes:
Z Set if (Rd) = 0, otherwise cleared
N Set if (Rd) < 0, otherwise cleared
C Unaffected
O Unaffected

Load Register(s) From Memory

The following instructions are used to load byte, word, double-word, triple-word and file-size operands into register(s). A memory byte operand is placed in the lower byte position of a register and the upper byte position of the register is cleared to zero. All other memory operands replace the entire contents of the number of registers affected.

Special Condition Code values are set to classify byte operands whose 8-bit codes are ASCII special characters. Appropriate Condition Codes are set for all other operands that are zero or negative.

LBX,r,x Load Byte from Memory (Byte-Indexed)

0 7	8 11	12 15
AE	Rd	Rx

(VEBA) → $(Rd)_{8\text{-}15}$

0's → $(Rd)_{0\text{-}7}$

Register Rx

0 7 8 11 12 15
Virtual Base Address

Register RxV1

0	1 14	15
S	Word Displacement	0 1

The contents of memory byte location VEBA replace the right byte contained in register Rd. Zeroes replace the left byte contained in register Rd.

Affected: (Rd)

Condition Codes:

1000 If byte is (
1001 If byte is)
1010 If byte is *
1011 If byte is +
1100 If byte is ,
1101 If byte is -

1110 If byte is .
1111 If byte is /
0000 If byte is alphabetic
0001 No special code
0011 If byte is space
0101 If byte is numeric

LDM*,r,x A Load Register from Memory

0 7	8 11	12	15
E5	Rd	I	Rxxx
Virtual Operand Address			

(VEA) → (Rd)

The contents of memory location VEA replace the contents of register Rd.

Affected: (Rd)

Condition Codes:
Z Set if all registers loaded = 0, otherwise cleared
N Set if bit 0 of Rd = 1, otherwise cleared
C Unaffected
O Unaffected

LDI,r V — Load Register from Memory (Immediate)

0 … 7	8 … 11	12 … 15
ED	Rd	0
Immediate Operand		

((PR)+1)→(Rd)

The contents of immediate memory location VEA replace the contents of register Rd.

Affected: (Rd)

Condition Codes:
Z Set if all registers loaded = 0, otherwise cleared
N Set if bit 0 of Rd = 1, otherwise cleared
C Unaffected
O Unaffected

LDS,r,d — Load Register from Memory (Short-Displaced)

0 … 7	8 … 11	12 … 15
F5	Rd	D

((R1)+D)→(Rd)

The contents of short-displaced memory location VEA replace the contents of register Rd.

Affected: (Rd)

Condition Codes
Z Set if all registers loaded = 0, otherwise cleared
N Set if bit 0 of Rd = 1, otherwise cleared
C Unaffected
O Unaffected

LDX,r,x — Load Register from Memory (Short-Indexed)

0 … 7	8 … 11	12 … 15
FD	Rd	Rx

((Rx))→(Rd)

The contents of short-indexed memory location VEA replace the contents of register Rd.

Affected: (Rd)

Condition Codes:
Z Set if all registers loaded = 0, otherwise cleared
N Set if bit 0 of Rd = 1, otherwise cleared
C Unaffected
O Unaffected

LDMD*,r,x A — Load Double-Register from Memory Doubleword

0 … 3 4 … 7	8 … 11		12	… 15
CD	Rd	$\overline{1}$	I	Rxx
Virtual Operand Address				

(VEA:VEA+1)→(Rd:RdV1)

The contents of memory locations VEA: VEA+1 replace the contents of registers Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:
Z Set if all registers loaded = 0, otherwise cleared
N Set if bit 0 of Rd = 1, otherwise cleared
C Unaffected
O Unaffected

LDMT*,r,x A Load Triple-Register from Memory Tripleword

0 7	8 9	10	11	12	15
88	Rd	1	1	I	Rxx

(VEA:VEA+1 VEA+2)→ (Rd:RdV1:RdV2)

VOA

The contents of memory locations VEA:VEA+1:VEA+2 replace the contents of registers Rd:RdV1:RdV2.

Affected: (Rd:RdV1:RdV2)

Condition Codes:
Z Set if all registers load = 0, otherwise cleared
N Set if bit 0 of Rd = 1, otherwise cleared
O Unaffected
C Unaffected

LDXD,r,x Load Double-Register from Memory Doubleword (Short-Indexed)

0 7	8	11	12 15
8D	Rd	1	Rx

((Rx):(Rx)+1)→(Rd:RdV1)

The contents of short-indexed memory locations VEA: VEA+1 replace the contents of registers Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:
Z Set if all registers loaded = 0, otherwise cleared
N Set if bit 0 of Rd = 1, otherwise cleared
C Unaffected
O Unaffected

LDXT,r,x Load Triple-Register From Memory Tripleword (Short-Indexed)

0 7	8 9	10 11	12 15
88	Rd	01	Rx

((Rx):(Rx)+1:(Rx)+2)→(Rd:RdV1:RdV2)

The contents of short-indexed memory locations VEA: VEA+1:VEA+2 replace the contents of registers Rd:RdV1:RdV2.

Affected: (Rd:RdV1:RdV2)

Condition Codes:
Z Set if all registers loaded = 0, otherwise cleared
N Set if bit 0 of Rd = 1, otherwise cleared
C Unaffected
O Unaffected

LFM*,r,x A Load File From Memory

0 7	8 11	12	15
A4	Rd	I	Rxx
Virtual Operand Address			

(VEA,VEA+1,.. .. . .VEA+N-1)→(Rd,Rd+1, . . . R7 if Rd<7)
R15 if Rd>7)

The contents of from one to eight consecutive memory locations starting with memory location VEA replace the contents of register Rd through R7 if Rd⩽7, or register Rd through R15 if Rd >7.

Affected: (Rd through R7 or R15)

Condition Codes:
Z Set if all registers loaded = 0, otherwise cleared
N Set if bit 0 of Rd = 1, otherwise cleared
C Unaffected
O Unaffected

LFS,r,d — Load Fiie From Memory (Short-Displaced)

0 7	8 11	12 15
B4	Rd	D

((R1)+D,(R1)+D+1, . . . (R1)+D+N-1) → (Rd,Rd+1, . . .
. . . R7 if Rd < 7)
. . . R15 if Rd > 7)

The contents of from one to eight consecutive memory locations starting with short-displaced memory location VEA replace the contents of registers Rd through R7, if Rd ⩽ 7, or register Rd through R15 if Rd > 7.

Affected: (Rd through R7 or R15)

Condition Codes:
Z Set if all registers loaded = 0, otherwise cleared
N Set if bit 0 of Rd = 1, otherwise cleared
C Unaffected
O Unaffected

LFX,r,x — Load File From Memory (Short-Indexed

0 7	8 11	12 15
BC	Rd	Rx

((Rx),(Rx)+1, . . .(Rx)+N-1)→(Rd,Rd+1, . . .
. . . R7 if Rd < 7
. . . R15 if Rd > 7)

The contents of from one to eight consecutive memory locations starting with short-indexed memory location VEA replace the contents of registers Rd through R7 if Rd ⩽ 7, or register Rd through R15 if Rd > 7.

Affected: (Rd through R7 or R15)

Condition Codes:
Z Set if all registers loaded = 0, otherwise cleared
N Set if bit 0 of Rd = 1, otherwise cleared
C Unaffected
O Unaffected

Clear Memory

The following instructions are used to reset to zero a word(s) in memory.

These instructions do not set Condition Codes.

CLM Clear Memory

0 7	8 10	11	12	13 15
CE	0	0	1	Rxx
Virtual Operand Address				

0→(VEA)

The contents of memory location VEA are cleared to zero. This instruction must not alter the 8 words following its own location - i.e., (PR)+2 to (PR)+9.

Affected: (VEA)

Condition Codes: Not affected

CLMD Clear Memory Double Word

0 7	8 10	11	12	13 15
CE	1	0	1	Rxx
Virtual Operand Address				

0→(VEA), (VEA+1)

The contents of memory locations VEA,VEA+1 are cleared to zero. The instruction must not alter the 8 words following its own location, i.e., (PR)+2 to (PR)+9.

Affected: (VEA), (VEA+1)

Condition Codes: Not affected

Store Register(s) in Memory

The following instructions are used to store byte, word, double-word, triple-word and file-size operands from registers to memory. The byte operand in the lower byte position of a register replaces a memory byte location. All other register operands replace the entire contents of the number of memory word locations affected.

These instructions do not set Condition Codes.

SBX,s,x Store Byte in Memory (Byte-(Indexed)

0 7	8 11	12 15
AF	Rs	Rx

$(Rs)_{8\text{-}15} \longrightarrow (VEBA)$

0 15
Virtual Base Address

Rx

0	1 14	15
S	Word Displacement	0 1

RxV1

The right byte in register Rs replaces the contents of memory byte location VEBA. The other byte in the memory word is not affected. Refer to Chapter 8 for a description of byte addressing.

Affected: (VEBA)

Condition
Codes: Unaffected

STM*,s,x A Store Register in Memory

0 7	8 11		12 15
E6	Rs	I	Rxx
Virtual Operand Address			

(Rs) ⟶ (VEA)

The contents of register Rs replace the contents of memory location VEA.

Affected: (VEA)

Condition
Codes: Unaffected

STI,s V Store Register in Memory (Immediate)

0 7	8 11	12 15
EE	Rs	0
Immediate Operand		

(Rs) ⟶ ((PR)+1)

The contents of register Rs replace the contents of immediate memory location VEA.

Affected: (VEA)

Condition
Codes: Unaffected

STS,s,d Store Register in Memory (Short-Displaced)

0 7	8 11	12 15
F6	Rs	D

(Rs) ⟶ ((R1)+D)

The contents of register Rs replace the contents of short-displaced memory location VEA.

Affected: (VEA)

Condition
Codes: Unaffected

STX,s,x Store Register in Memory (Short-Indexed)

0 7	8 11	12 15
FE	Rs	Rx

(Rs)→((Rx))

The contents of register Rs replace the contents of short-indexed memory location VEA.

Affected: (VEA)

Condition Codes: Unaffected

STMD*,s,x A Store Double-Register in Memory Doubleword

<table>
<tr><th>0 7</th><th>8 11</th><th>12</th><th></th><th>15</th></tr>
<tr><td>CE</td><td>Rs</td><td>1</td><td>I</td><td>Rxx</td></tr>
<tr><td colspan="5">Virtual Operand Address</td></tr>
</table>

(Rs:RsV1)→(VEA:VEA+1)

The contents of register Rs:RsV1 replace the contents of memory locations VEA:VEA+1. This instruction must not alter the 8 words following its own location, i.e., (PR)+2 to (PR)+9.

Affected: (VEA:VEA+1)

Condition Codes: Unaffected

STMT*,r,x A Store Triple-Register in Memory Tripleword

<table>
<tr><th>0 7</th><th>8 9</th><th>10</th><th>11</th><th>12</th><th>13 15</th></tr>
<tr><td>88</td><td>Rs</td><td>1</td><td>0</td><td>I</td><td>Rxx</td></tr>
<tr><td colspan="6">Virtual Operand Address</td></tr>
</table>

(Rs:RsV1:RsV2)→(VEA: VEA+1: VEA+2)

The contents of registers Rs:RsV1:RsV2 replace the contents of memory locations VEA:VEA+1:VEA+2. This instruction must not alter the 8 words following in its own location i.e., (PR)+2 to (PR)+9.

Affected: (VEA:VEA+1:VEA+2)

Condition Codes: Unaffected

STXD,s,x Store Double-Register in Memory Doubleword (Short-Indexed)

0 7	8 10	11	12 15
8E	Rs	1	Rx

(Rs:RsV1)→((Rx):Rx)+1)

The contents of registers Rs:RsV1 replace the contents of short-indexed memory locations VEA:VEA+1. This instruction must not alter the 8 words following its own location i.e., (PR)+1 to (PR)+8.

Affected: (VEA:VEA+1)

Condition Codes: Unaffected

STXT,s,x — Store Triple-Register in Memory Tripleword (Short-Indexed)

0 7 8 10 11 12 15

88	Rs	0 0	Rx

(Rs:RsV1:RsV2)→((Rx):(Rx)+1:(Rx)+2)

The contents of registers Rs:RsV1:RsV2 replace the contents of short-indexed memory locations VEA:VEA+1: VEA+2. This instruction must not alter the 8 words following its own location i.e., (PR)+1 to (PR)+8.

Affected: (VEA,VEA+1,VEA+2)

Condition Codes: Unaffected

SFM*,s,x A — Store File in Memory

11 12

0 3 4 7 8 ~~10 11 12~~ 15

A5	Rs	I	Rxx
Virtual Operand Address			

(Rs,Rs+1,... R7 if Rs < or = to 7)→
... R15 if Rs > 7)
(VEA,VEA+1,... VEA+N-1)

The contents of registers Rs through R7 if Rs ≤ 7, or registers Rs through R15 if Rs > 7, replace the contents of from one to eight consecutive memory locations starting with memory location VEA. This instruction must not alter the 8 words following its own location i.e., (PR)+2 to (PR)+9.

Affected: (VEA, VEA+1,... VEA+N-1)

Condition Codes: Unaffected

SFS,s,d — Store File in Memory (Short-Displaced)

0 7 8 11 12 15

B5	Rs	D

(Rs,Rs+1,... R7 if Rs< or = to 7)→((R1)+D,(R1)+D+1,...
R15 if Rs>7) →(Rs)+D+N-1)

The contents of registers Rs through R7 if Rs ≤ 7, or registers Rs through R15 if Rs > 7, replace the contents of from one to eight consecutive memory locations starting with short-displaced memory location VEA. This instruction must not alter the 8 words following its own location i.e., (PR)+1 to (PR)+8.

Affected: (VEA, VEA+1,... VEA+N-1)

Condition Codes: Unaffected

SFX,s,x — Store File in Memory (Short-Indexed)

0 7 8 11 12 15

BD	Rs	Rx

(Rs,Rs+1,... R7 if Rs ≤ 7)→((Rx),(Rx)+1,... (Rx)+N-1)
... R15 if Rs > 7)

The contents of registers Rs through R7 if s ≤ 7, or registers Rs through R15 if s > 7, replace the contents of from one to eight consecutive memory locations starting with short-indexed memory location VEA. This instruction must not alter the 8 words following its own location i.e., (PR)+1 to (PR)+8.

Affected: (VEA, VEA+1,... VEA+N-1)

Condition Codes: Unaffected

Push Register(s) into Memory LIFO Stack

The following instruction is used to allocate space toward and store registers into, the low end of a last-in-first-out memory stack. The limits and current position in the stack are defined in a separate control table in memory.

The number of words (0-255) allocated to the stack are sub-tracted from the current position in the stack to extend its addressable space. From 1 to 16 registers (with wraparound) then are stored starting at the new current position in the stack. If the number of registers transferred is less than the number of words allocated, the remaining higher allocated memory locations are not changed.

Stack overflow occurs if the new current position in the stack becomes lower than the low end of the stack. The normal operation then aborts, appropriate Condition Codes are set, register R1 is saved then loaded with the current Program Counter, and a branch is executed to a user overflow exit address.

If the "push" is executed at interrupt levels, the first instruction of the user exit (which is uninterruptable) should disable further interrupts to allow it to access the saved value of register R1, which otherwise could become changed.

PSM*,r,x A,NR,NW — Push Register(s) Into Memory LIFO Stack

0 - 7	*8 - 10*	*11*	*12*	*13 - 15*
BB	Rs	1	I	Rxx
Virtual SPT + 1 Address				

0 - 3	*4 - 7*	*8 - 15*
0	NR-1	NW

CSP-NW=CSP=VEA → (SPT+1)

(Rs,Rs+1, . . . Rs+NR-1*) → (VEA,VEA+1, . . . VEA+NR-1)

If CSP-NW < LSA: (R1) → (SPT+4)
(PR) → (R1)
(SPT+3) → (PR)

[*-16, if wraparound]

0 — 15	
Lowstack Address (LSA)	SPT
Current STK PTR (CSP)	SPT+1
High STK ADD.+1 (HSA)	SPT+2
Overflow/Underflow Address (OV)	SPT+3
Save R1 if Overflow/Underflow	SPT+4

Standard addressing locates the Stack Pointer Table plus 1 (SPT+1) location in memory. The current stack point (CSP) minus the number of words allocated (NW) forms the new CSP and becomes the VEA, replacing the contents of SPT+1. The contents of the specified number of registers (NR) starting with register Rs replace the contents of NR locations starting at the VEA. If Rs+NR > 16, register wraparound will occur. If CSP-NW < LSA, overflow has occureed, the contents of register R1 replace the contents of SPT+4, the contents of the Program Register (PR) replace the contents of register R1, and the contents of SPT+3 replace the contents of the Program Register.

This instruction must not alter the 8 words following its location, i.e., (PR)+3 to (PR)+10.

Affected: (SPT+1), (VEA,VEA+1, . . . VEA+NR-1)
If overflow: (SPT+4), (R1), (PR)

Condition Codes: Unaffected unless overflow occurs.

If overflow occurs, the result of (CSP-NW)-LSA causes:

- Z Set if result = 0, otherwise cleared
- N Set if result <0, otherwise cleared
- C Set if carry out of most significant bit, otherwise cleared
- O Set if operands were of opposite sign and the sign of the subtrahend was the same as the sign of the result, otherwise cleared

Pull Register(s) from Memory LIFO Stack

The following instruction is used to load registers and deallocate space from the low end of a last-in-first-out memory stack. The limits and current position in the stack are defined in a separate control table in memory.

From 1 to 16 registers (with wraparound) are loaded starting from the current position in the stack. The number of words (0-255) deallocated from the stack are added to the current position in the stack to decrease its addressable space. The number of words deallocated may be greater than the number of registers transferred, making the remaining higher deallocated memory locations equally unaddressable.

Stack underflow occurs if the new current position in the stack becomes higher than the high end of the stack. The normal operation then aborts, appropriate Condition Codes are set, register R1 is saved then loaded with the current Program Counter, and a branch is executed to a user underflow exit address.

If the "pull" is executed at interrupt levels, the first instruction of the user exit (which is uninterrupted), should disable further interrupts to allow it to access the saved value of register R1 which otherwise could become changed.

PLM*,r,x A,NR,NW — Pull Register(s) From Memory LIFO Stack

0 3 4 7 8 11 12 15

BA	Rd	I	Rxx
Virtual SPT+1 Address			
0	NR-1	NW	

```
CSP=VEA
)VEA,VEA+1,... VEA+NR-1)→(Rd,Rd+1,... Rd+NR-1*)
            CSP+NW=CSP  →(SPT+1)

If CSP+NW > HSA:(R1)    →(SPT+4)
                (PR)    →(R1)
              (SPT+3)   →(PR)
```

[*-16, if wraparound]

0 15

Low STK Address (LSA)	SPT
Current STK ADD. (CSP)	SPT+1
High STK ADD.+1 (HSA)	SPT+2
Overflow/Underflow Address (OV)	SPT+3
Save R1 if Overflow/Underflow	SPT+4

Standard addressing locates the Stack Pointer Table plus 1 (SPT+1) location in memory. The Current Stack Pointer (CSP) becomes the VEA. The contents of NR locations starting at the VEA replace the contents of the specified number of registers (NR) starting with register Rd.

(If RD+NR > 16, register wraparound will occur.)

The Current Stack Pointer (CSP) plus the number of words (NW) deallocated forms the new CSP and replaces the contents of SPT+1. If CSP+NW > HSA, underflow has occured, the contents of register R1 replace the contents of SPT+4, the contents of the Program Register (PR) replace the contents of register R1, and the contents of SPT+3 replace the contents of the Program Register.

Affected: (Rd,Rd+1,... Rd+NR-1, (SPT+1)
If underflow: (SPT+4), (R1), (PR)

Condition Codes: If normal operation:

Z	Set if all words pulled = 0, otherwise cleared
N	Set if first word pulled has bit 0 = 1, otherwise cleared
C	Cleared
O	Cleared

If underflow occurs, the result of (CSP+NW)-HSA causes the same conditions as PSM overflow

Move Block Between Virtual and Extended Memory

The following instructions transfer flocks of up to 64K words between virtually addressed memory and extended memory or between two areas of virtual memory. They are used to move blocks of data, input from an I/O device into virtually mapped memory, to extended memory, and to move required data blocks or tasks into a virtually mapped area.

MBEV,r — Move Block From Extended To Virtual Memory

0 7	8 9	10 11	12 15
OE	R	00	9

(EA) . . . (EA+TC-1) → (VA) . . . (VA+TC-1)

The value R specifies a quadruple-register whose format will be as below:

0 10	11 15	
Transfer Count		R
Virtual Address		RV1
	EMA	RV2
Actual Address		RV3

This instruction transfers up to 64K words of data from the task's extended addressing space to the task's Virtual operand space. The transfer count is stored in an implied 17 bit two's complement form, where the MSB is implied to be one. This allows transfer counts in the range 1-65,536. Transfers take place in blocks of up to four words at a time, and continue until the transfer count is incremented to zero. The instruction is interruptable after each block transfer, and the register parameters are updated as the result of each block transfer.

If a system protect violation occurs, the register pointers will point to the block that caused the violation.

Virtual address "rollover" is allowed. Extended address calculations use the Extended Memory Control Register (EMCR) for address biasing and protection, but the MBEV instruction should not be preceded by an EXMA instruction. This instruction must not alter the 8 words following its own location, i.e., (PR)+1 to (PR)+8.

Affected: Virtual Address locations VA to VA+TC-1

Condition Codes:
N Reset upon exit
Z Set upon exit
O,C Not affected

MBVE,r — Move Block From Virtual To Extended Memory

0 7	8 9	10 11	12 15
OE	R	00	8

(VA) . . . (VA+TC-1) → (EA) . . . (EA+TC-1)

The value R specifies a quadruple register whose format will be as shown below:

0 10	11 15	
Transfer Count		R
Virtual Address		RV1
	EMA	RV2
Actual Address		RV3

The instruction transfers up to 64K words of data from the tasks virtual operand space to the task's extended addressing space. The transfer count is stored in an implied 17 bit two's complement form, where the MSB is implied to be one. This allows transfer counts in the range 1-65,536. Transfers take place in blocks of up to four words at a time, and continue until the transfer count is incremented to zero. The instruction is interruptable after each block transfer, and the register parameters are updated as the result of each block transfer.

If a system protect violation occurs, the register pointers will point to the block that cuased the violation.

Virtual address "rollover" is allowed. Extended address calculations use the Extended Memory Control Register (EMCR) for biasing and protection, but the MBVE instruction should not be preceded by an EXMA instruction. This instruction must not alter the 8 words following its own location, i.e., (PR)+1 to (PR)+8.

Affected: Extended Memory Location EMA:MA to EMA:MA+TC-1

Condition Codes:
N Reset upon exit
Z Set upon exit
O,C Not affected

MBVV,r,s — Move Virtual Block To Virtual Block

0 7	8 11	12 15
OE	Rs	7

(SA) . . . (SA+TC-1) → (DA) . . . (DA+TC-1)

Transfer Count	Rs
Source Address	RsV1
Destination Address	RsV2

The contents of virtual memory location (RsV1) through (RsV1) + (Rs)-1.are written into virtual memory locations (RsV2) through (RsV2) + (Rs)-1. The contents of the source block are not affected.

The transfer count is stored in an implied 17 bit two's complement form, where the MSB is implied to be one. This allows transfer counts in the range 1-65,536. Transfers take place in blocks of up to four words at a time, and continue until the transfer count is incremented to zero. The instruction is interruptable after each block transfer, and the register parameters are updated as the result of each block transfer.

If a system protect violation occurs, the register pointers will point to the block that caused the violation.

Virtual address "rollover" is allowed.

This instruction must not alter the 8 words following its own location, i.e., (PR)+1 to (PR)+8.

Affected: (VEA) – (VEA+TC)-1
(Rs)

Condition Codes:
N Reset
Z Set
O,C Not affected

Fixed Point Arithmetic Instructions

This instruction class provides the capability to perform algebraic add, subtract, multiply, divide, two's complement, and extend sign operations on bit, word and multi-word fixed point integer operands.

Bit, word, and doubleword add, subtract, multiply, and divide operations may take place in the general purpose registers using bit constants or source operands in the registers or memory. Bit and word addition also may take place in memory using bit constants or source operands in registers. Word, doubleword and quadruple-word two's complement and extend sign operations occur in registers.

The memory operations include most standard addressing modes. In addition, many of these operations provide an option for the program to branch when the result is nonzero. All of the instructions in this class set Condition Codes.

All operations assume fixed point integer operands. They also assume that operand register psotioning and unit length are compatible with those designated by the operation code.

Use of an odd destination register in word multiply or divide, or of destination register R2, R6, R10 or R14 in doubleword multiply or divide, will result in a loss of the most significant half of the product or all of the remainder. All arithmetic instructions, except those that multiply and extend sign, are capable of producing an overflow.

Add Operand to Register(s) (and Branch if Nonzero)

The following instructions add bit constants or fixed point word or doubleword operands contained in registers or memory to the General Purpose Registers. Bit or word operations in registers may be specified to branch if the result is not zero.

Overflow will occur when operands of like signs produce a result with the opposite sign.

Appropriate Condition Codes are set if the result is zero or negative, or to indicate adder carry out or overflow.

ABR,r,b Add Bit in Register

0 7	8 11	12 15
60	Rd	b

$(Rd)+2^{15-b} \rightarrow (Rd)$

The contents of register Rd are incremented by 2^{15-b}. The sum is stored in register Rd.

Affected: (Rd)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set if result <0, otherwise cleared
- C Set if carry out of most significant bit, otherwise cleared
- O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

ABRB,r,b B Add Bit in Register and Branch if Nonzero

0 7	8 11	12 15
70	Rd	b
Virtual Branch Address		

$(Rd)+2^{15-b} \rightarrow (Rd)$
If result does not = 0, $((PR)+1) \rightarrow (PR)$
If result = 0, $(PR)+2 \rightarrow (PR)$

The contents of register Rd are incremented by 2^{15-b}. The sum is stored in register Rd. If the result is unequal to zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: (Rd)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set if result <0, otherwise cleared
- C Set if carry out of most significant bit, otherwise cleared
- O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

ADR,r,s Add Register to Register

0 7	8 11	12 15
68	Rd	Rs

$(Rs)+(Rd) \rightarrow (Rd)$

The contents of register Rs are algebraically added to the contents of register Rd. The sum is stored in register Rd.

Affected: (Rd)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set if result < 0, otherwise cleared
- C Set if carry out of most significant bit, otherwise cleared
- O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

ADRB,r,s B Add Register to Register and Branch if Nonzero

0 7	8 11	12 15
78	Rd	Rs
Virtual Branch Address		

$(Rs)+(Rd) \rightarrow (Rd)$
If result does not = 0, $((PR)+1) \rightarrow (PR)$
If result = 0, $(PR)+2 \rightarrow (PR)$

The contents of register Rs are algebraically added to the contents of register Rd. The sum is stored in register Rd. If the result does not equal zero, a branch is executed to the virtual branch address, otherwise, the next instruction in sequence is executed.

Affected: (Rd)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set if result <0, otherwise cleared
- C Set if carry out of most significant bit, otherwise cleared
- O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

ADRD,r,s (DAR) — Add Double-Register to Double-Register

0 7	8 11	12		15
C8	Rd	0	Rs	0

(Rs:RsV1)+(Rd:RdV1) → (Rd:RdV1)

The contents of registers Rs:RsV1 are algebraically added to the contents of registers Rd:RdV1. The sum replaces the contents of registers Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set if result < 0, otherwise cleared
- C Set if carry out of most significant bit, otherwise cleared
- O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

ADM*,r,x A — Add Memory to Register

0 7	8 11	12	15
E0	Rd	I	Rxx
Virtual Operand Address			

(VEA)+(Rd) → (Rd)

The contents of memory location VEA are algebraically added to the contents of register Rd. The sum is stored in register Rd.

Affected: (Rd)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set if result<0, otherwise cleared
- C Set if carry out of most significant bit, otherwise cleared
- O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

ADI,r V — Add Memory (Immediate) to Register

0 7	8 11	12 15
E8	Rd	0
Immediate Operand		

((PR)+1)+(Rd) → (Rd)

The contents of immediate memory location VEA are algebraically added to the contents of register Rd. The sum is stored in register Rd.

Affected: (Rd)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set if result < 0, otherwise cleared
- C Set if carry out of most significant bit, otherwise cleared
- O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

ADS,r,d — Add Memory (Short-Displaced) to Register

0 7	8 11	12 15
F0	Rd	D

((R1)+D)+(Rd) → (Rd)

The contents of short-displaced memory location VEA are algebraically added to the contents of register Rd. The sum is stored in register Rd.

Affected: (Rd)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set if result =<, otherwise cleared
- C Set if carry out of most significant bit, otherwise cleared
- O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

ADX,r,x Add Memory (Short-Indexed) To Register

0 — 7	8 — 11	12 — 15
F8	Rd	Rx

((Rx))+(Rd) → (Rd)

The contents of short-indexed memory location VEA are algebraically added to the contents of register Rd. The sum is stored in register Rd.

Affected: (Rd)

Condition Codes:

Z Set if result = 0, otherwise cleared

N Set if result< 0, otherwise cleared

C Set if carry out of most significant bit, otherwise cleared

O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

ADMD*,r,x A Add Memory Doubleword To Double-Register

0 — 7	8 — 11	12		— 15
C8	Rd	1	I	Rxx
Virtual Operand Address				

(VEA:VEA+1)+(Rd:RdV1) → (Rd:RdV1)

The contents of memory location VEA:VEA+1 are algebraically added to the contents of registers Rd:RdV1. The sum replaces the contents of registers Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:

Z Set if result = 0, otherwise cleared

N Set if result< 0, otherwise cleared

C Set if carry out of most significant bit, otherwise cleared

O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

Add Operand to Memory (and Branch if Nonzero)

The following instructions add bit constants and fixed point word operands contained in registers to memory. All operations may be specified to branch if the result is not zero.

Overflow will occur when operands of like sign produce a result with the opposite sign.

Appropriate Condition Codes are set if the result is zero or negative, or to indicate adder carry out or overflow.

ABMM*,b,x A — Add Bit in Memory

0 7	8 11	12	15
80	b	I	Rx
Virtual Operand Address			

$(VEA)+2^{15-b} \rightarrow (VEA)$

The contents of memory location VEA are incremented by 2^{15-b}. The result is stored in memory location VEA.

Affected: (VEA)

Condition Codes:

- Z Set if result = 0 , otherwise cleared
- N Set if result < 0, otherwise cleared
- C Set if carry out of most significant bit, otherwise cleared
- O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

ABMB*,b,x A,B — Add Bit in Memory and Branch if Nonzero

0 7	8 11	12	15
84	b	I	Rxx
Virtual Operand Address			
Virtual Branch Address			

$(VEA)+2^{15-b} \rightarrow (VEA)$

If Result does not = 0, $((PR)+2) \rightarrow (PR)$

If Result = 0, $(PR)+3 \rightarrow (PR)$

The contents of memory location VEA are incremented by 2^{15-b}. The result is stored in memory location VEA. If the result is unequal to zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: (VEA)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set if result < 0, otherwise cleared
- C Set if carry out of most significant bit, otherwise cleared
- O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

ABSM,b,d — Add Bit in Memory (Short-Displaced)

0 7	8 11	12 15
90	b	D

$((R1)+D)+2^{15-b} \rightarrow ((R1)+D)$

The contents of short-displaced memory location VEA are incremented by 2^{15-b}. The result is stored in memory location VEA.

Affected: (VEA)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set if result < 0, otherwise cleared
- C Set if carry out of most significant bit, otherwise cleared
- O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

ABSB,b,d B — Add Bit in Memory (Short-Displaced) and Branch if Nonzero

0 7	8 11	12 15
94	b	D
Virtual Branch Address		

$((R1)+D+2^{15-b} \rightarrow ((R2)+D)$
If Result does not = 0,((PR)+1) → (PR)
If Result = 0, (PR)+2 → (PR)

The contents of short-displaced memory location VEA are incremented by 2^{15-b}. The result is stored in memory location VEA. If the result is unequal to zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: (VEA)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set if result < 0, otherwise cleared
- C Set if carry out of most significant bit, otherwise cleared
- O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

ABXM,b,x — Add Bit in Memory (Short-Indexed)

0 7	8 11	12 15
98	b	Rx

$((Rx))+2^{15-b} \rightarrow ((Rx))$

The contents of short-indexed memory location VEA are incremented by 2^{15-b}. The result is stored in memory location VEA.

Affected: (VEA)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set if result < 0, otherwise cleared
- C Set if carry out of most significant bit, otherwise cleared
- O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

ABXB,b,x B — Add Bit in Memory (Short-Indexed) and Branch if Nonzero

0 7	8 11	12 15
9C	b	Rx
Virtual Branch Address		

$((Rx))+2^{15-b} \rightarrow ((Rx))$
If Result does not = 0,((PR)+1) → (PR)
If Result = 0,(PR)+2 → (PR)

The contents of short-indexed memory location VEA are incremented by 2^{15-b}. The result is stored in memory location VEA. If the result is unequal to zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed

Affected: (VEA)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set if result < 0, otherwise cleared
- C Set if carry out of most significant bit, otherwise cleared
- O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

ADMM*,s,x A Add Register to Memory

0 7	8 11	12	15
C0	Rs	I	Rxx
Virtual Operand Address			

(Rs)+(VEA) → (VEA)

The contents of register Rs are algebraically added to the contents of memory location VEA. The sum is stored in memory location VEA.

Affected: (VEA)

Condition Codes:

Z Set if result = 0, otherwise cleared
N Set if result < 0, otherwise cleared
C Set if carry out of most significant bit, otherwise cleared
O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

ADMB*,s,x A,B Add Register to Memory and Branch if Nonzero

0 7	8 11	12	15
C4	Rs	I	Rxx
Virtual Operand Address			
Virtual Branch Address			

(Rs)+(VEA) → (VEA)
If Result does not = 0,((PR)+2) → (PR)
If Result = 0,(PR)+3 → (PR)

The contents of register Rs are algebraically added to the contents of memory location VEA. The sum is stored in memory location VEA. If the result does not equal zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: (VEA)

Condition Codes:

Z Set if result = 0, otherwise cleared
N Set if result < 0, otherwise cleeared
C Set if carry out of most significant bit, otherwise cleared
O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

ADSM,s,d Add Register to Memory (Short-Displaced)

0 7	8 11	12 15
D0	Rs	D

(Rs)+((R1)+D) → ((R1)+D)

The contents of register Rs are algebraically added to the contents of short-displaced memory location VEA. The result is stored in the memory location VEA.

Affected: (VEA)

Condition Codes:

Z Set if result = 0, otherwise cleared
N Set if result < 0, otherwise cleared
C Set if carry out of most significant bit, otherwise cleared
O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

ADSB,s,d B Add Register to Memory (Short-Displaced) and Branch if Nonzero

0 7	8 11	12 15
D4	Rs	D
Virtual Branch Address		

(Rs+((R1)+D) → ((R1)+D)
If Result does not = 0,((PR)+1) → (PR)
If Result = 0,(PR)+2 → (PR)

The contents of register Rs are algebraically added to the contents of short-displaced memory location VEA. The sum is stored in memory location VEA. If the result does not equal zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: (VEA)

Condition Codes:

Z Set if result = 0, otherwise cleared
N Set if result < 0, otherwise cleared
C Set if carry out of most significant bit, otherwise cleared
O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

ADXM,s,x — Add Register to Memory (Short-Indexed)

0 7	8 11	12 15
D8	Rs	Rx

(Rs)+((Rx)) → ((Rx))

The contents of register Rs are algebraically added to the contents of short-indexed memory location VEA. The sum is stored in memory location VEA.

Affected: (VEA)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set if result < 0, otherwise cleared
- C Set if carry out of most significant bit, otherwise cleared
- O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

ADXB,s,x B — Add Register to Memory (Short-Indexed) and Branch if Nonzero

0 7	8 11	12 15
DC	Rs	Rx
Virtual Branch Address		

(Rs)+((Rx)) → ((Rx))
If Result does not = 0,((PR)+1) → (PR)
If Result = 0,(PR)+2 → (PR)

The contents of register Rs are algebraically added to the contents of short-indexed memory location VEA. The sum is stored in memory location VEA. If the result does not equal zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: (VEA)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set if result < 0, otherwise cleared
- C Set if carry out of most significant bit, otherwise cleared
- O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

Subtract Operand from Register(s) (and Branch if Nonzero)

The following instructions subtract bit constants, and fixed point word or doubleword operands contained in registers or memory, from the General Purpose Registers. Bit or word operations in registers may be specified to branch if the result is not zero.

The result is the algebraic difference of the specified operands, but is computed as the sum of the destination operand, the ones complement of the source oeprand and the constant one. Adder carry then is the complement of the algebraic subtraction.

Overflow will occur when operands of opposite signs produce a result with a different sign than the destination operand. Appropriate Condition Codes are set if the result is zero or negative, or to indicate adder carry out or overflow

SBR,r,b — Subtract Bit in Register

0 7	8 11	12 15
61	Rd	b

$(Rd)-2^{15-b} \rightarrow (Rd)$

The contents of register Rd are decremented by one in bit position b. The result is stored in register Rd.

Affected: (Rd)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result, otherwise cleared
- C Set if carry out of most significant bit
- O Set if operands were of opposite signs and the sign of the subtrahend was the same as the sign of the result, otherwise cleared

SBRB,r,b B — Subtract Bit in Register and Branch if Nonzero

0 7	8 11	12 15
71	Rd	b
Virtual Branch Address		

$(Rd)-2^{15-b} \rightarrow (Rd)$
If Result does not = 0, $((PR)+1) \rightarrow (PR)$
If Result = 0, $(PR)+2 \rightarrow (PR)$

The contents of register Rd are decremented by one in bit position b. The result is stored in register Rd. If the result is unequal to zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: (Rd)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result, otherwise cleared
- C Set if carry out of most significant bit
- O Set if operands were of opposite signs and the sign of the subtrahend was the same as the sign of the result, otherwise cleared

SUR,r,s — Subtract Register From Register

0 7	8 11	12 15
69	Rd	Rs

(Rd)-(Rs) → (Rd)

The contents of register Rs are algebraically subtracted from the contents of register Rd. The difference is stored in register Rd.

Affected: (Rd)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result, otherwise cleared
- C Set if carry out of most significant bit
- O Set if operands were of opposite signs and the sign of the subtrahend was the same as the sign of the result, otherwise cleared

SURB,r,s B — Subtract Register From Register and Branch if Nonzero

0 7	8 11	12 15
79	Rd	Rs
Virtual Branch Address		

(Rd)-(Rs) → (Rd)
If Result does not = 0,((PR)+1) → (PR)
If Result = 0,(PR)+2 → (PR)

The contents of register Rs are algebraically subtracted from the contents of register Rd. The difference is stored in register Rd. If the result does not equal zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: (Rd)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result, otherwise cleared
- C Set if carry out of most significant bit
- O Set if operands were of opposite signs and the sign of the subtrahend was the same as the sign of the result, otherwise cleared

SURD,r,s — Subtract Double-Register From Double-Register

0 7	8	11	12	15
C9	Rd	0	Rs	0

(Rd:RdV1)-(Rs:RsV1) → (Rd:RdV1)

The contents of registers Rs:RsV1 are algebraically subtracted from the contents of registers Rd:RdV1. The difference replaces the contents of registers Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result, otherwise cleared
- C Set if carry out of most significant bit
- O Set if operands were of opposite signs and the sign of the subtrahend was the same as the sign of the result, otherwise cleared

SUM*,r,x A — Subtract Memory From Register

0 3 4 7	8 11	12	15
E1	Rd	I	Rxx
Virtual Operand Address			

(Rd)-(VEA) → (Rd)

The contents of memory location VEA are algebraically subtracted from the contents of register Rd. The difference is stored in register Rd.

Affected: (Rd)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result, otherwise cleared
- C Set if carry out of most significant bit
- O Set if operands were of opposite signs and the sign of the subtrahend was the same as the sign of the result, otherwise cleared

SUS,r,d — Subtract Memory (Short-Displaced) From Register

0 7	8 11	12 15
F1	Rd	D

(Rd)–((R1)+D) ⟶(Rd)

The contents of short-displaced memory location VEA are algebraically subtracted from the contents of register Rd. The difference is stored in register Rd.

Affected: (Rd)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result, otherwise cleared
- C Set if carry out of most significant bit
- O Set if operands were of opposite signs and the sign of the subtrahend was the same as the sign of the result, otherwise cleared

SUI,r V — Subtract Memory (Immediate) From Register

0 7	8 11	12 15
E9	Rd	0
Immediate Operand		

(Rd)–((PR)+1) ⟶(Rd)

The contents of immediate memory location VEA are algebraically subtracted from the contents of register Rd. The difference is stored in register Rd.

Affected: (Rd)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result, otherwise cleared
- C Set if carry out of most significant bit
- O Set if operands were of opposite signs and the sign of the subtrahend was the same as the sign of the result, otherwise cleared

SUX,r,x — Subtract Memory (Short-Indexed) From Register

0 7	8 11	12 15
F9	Rd	Rx

(Rd)–((Rx)) ⟶(Rd)

The contents of short-indexed memory location VEA are algebraically subtracted from the contents of register Rd. The difference is stored in register Rd.

Affected: (Rd)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result, otherwise cleared
- C Set if carry out of most significant bit
- O Set if operands were of opposite signs and the sign of the subtrahend was the same as the sign of the result, otherwise cleared

SUMD*,r,x A — Subtract Memory Doubleword From Double-Register

0 7	8 11	12		15
C9	Rd	1	I	Rxx
Virtual Operand Address				

(Rd:RdV1)–(VEA:VEA+1)⟶(Rd:RdV1)

The contents of memory location VEA:VEA+1 are algebraically subtracted from the contents of registers Rd:RdV1. The difference replaces the contents of registers Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result, otherwise cleared
- C Set if carry out of most significant bit
- O Set if operands were of opposite signs and the sign of the subtrahend was the same as the sign of the result, otherwise cleared

Multiply Register(s) by Operand

The following instructions multiply fixed point word or doubleword operands.contained in the general purpose registers by operands in registers or memory.

The multiply/divide instructions form a compatible set. The true relationship (A x B) /A = B is accomplished by appropriate and consistent positioning of operands and results, with the multiply/quotient and product/divident occupying common register positions.

In double-register multiply instructions, if Rd specifies quadruple-register R4, R8, or R12 (see figure 9-1), the doubleword multiplier is contained in RdV2:RdV3 and the signed quadruple-word product will be stored in Rd:RdV1:RdV2:RdV3. If Rd specifies double-register R2, R6, R10, or R14, and the product contains more than 32 significant bits, the most significant bits are lost.

Similarly, for double-register divide instructions (see the next instruction group) the dividend normally is located in quadruple-register R4, R8 or R12 (see figure 9-1), and the signed quotient is stored in RdV2:RdV3 with the remainder in Rd:RdV1. Placement of the dividend in double-register R2, R6, R10, or R14 will not yield valid results.

Single-register multiply/divide operations are performed similarly, with even numbered destination double-registers (see figure 9-2) containing single-word multiplier in RdV1 and double-word product, dividend and remainder/quotient in Rd:RdV1. Odd numbered destination single-registers (see figure 9-3) may contain a single-word multiplier and a 16-bit product. If the product contains more than 16 bits, the most significant bits are lost.

If the same register is specified for both source and destination, the product replaces the original operand and is its square.

The maximum values for the products of double-word and single-word operands are as follows:

Operand Signs	*Maximum Operands*		*Maximum Product*
Double Word:			
(+ X +)	$(2^{31}-1) \times (2^{31}-1)$	=	$2^{62}-2^{32}+2^{0}$
(+ X -)	$(2^{31}-1) \times -2^{31}$	=	$-(2^{62}-2^{31})$
(- X -)	$-2^{31} \times -2^{31}$	=	2^{62}

Where minus full scale = 1000 0000 0000 0000 0000 0000 0000 $0000\ 0000_2 = -2^{31}$

Single Word:			
(+ X +)	$(2^{15}-1) \times (2^{15}-1)$	=	$2^{30}-2^{16}+2^{0}$
(+ X -)	$(2^{15}-1) \times -2^{15}$	=	$-(2^{30}-2^{15})$
(- X -)	$-2^{15} \times -2^{15}$	=	2^{30}

Where minus full scale = $1000\ 0000\ 0000\ 0000_2 = -2^{15}$

Fixed Point Arithmetic Instructions

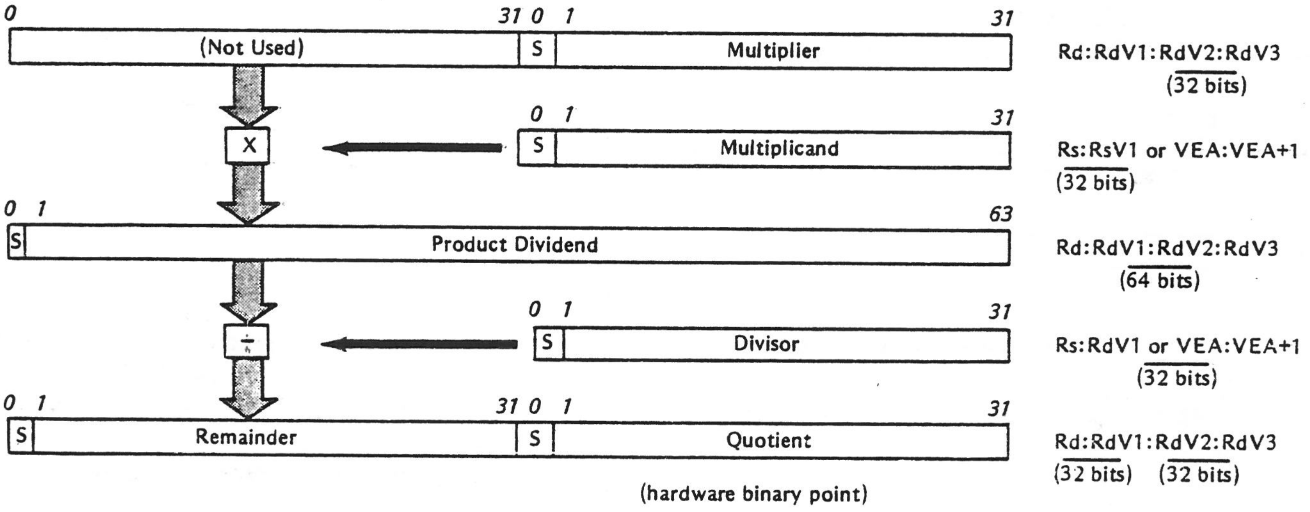
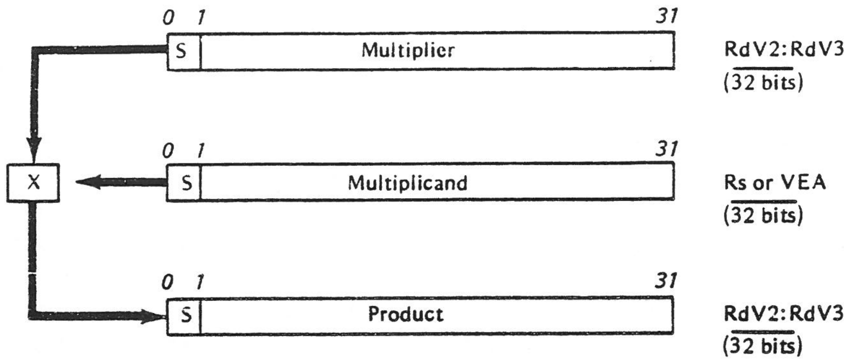


Figure 9-1. Full Precision Double-Register Multiply/Divide Using Quadruple-Registers R4, R8, or R12



Double Register Multiply Using R2, R6, R10 or R14 as Register Destination

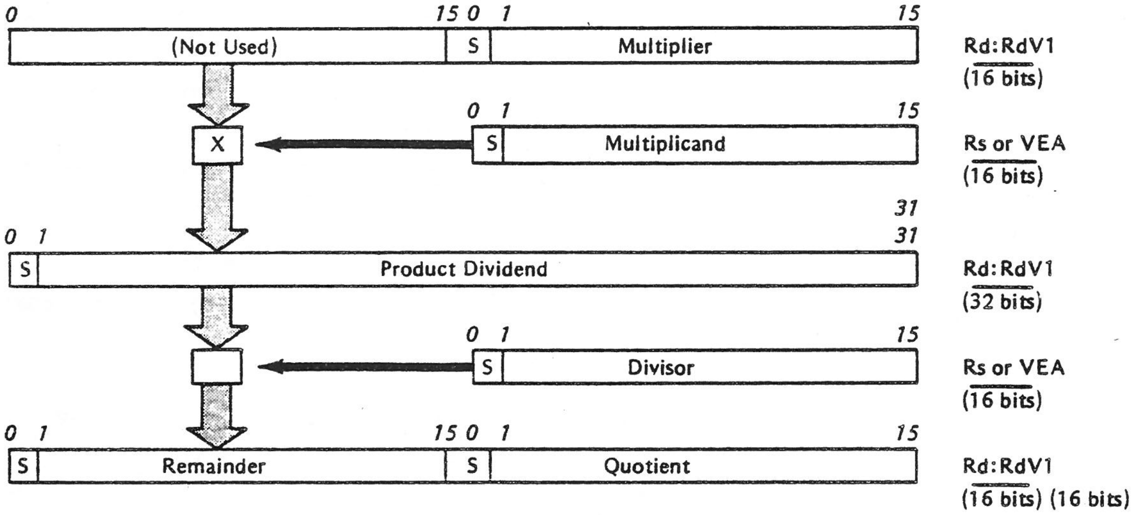


Figure 9-2. Single-Register Multiply/Divide Using Even Numbered Destination Double-Registers

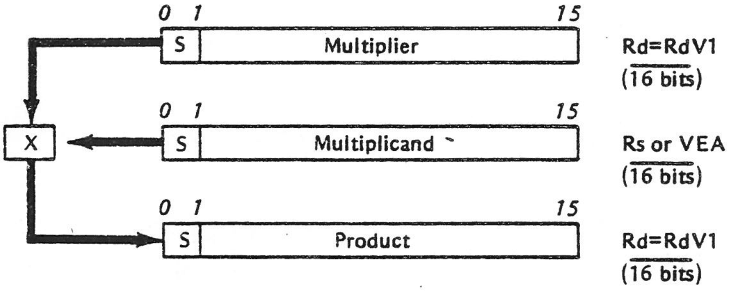


Figure 9-3. Single-Register Multiply Using Odd Numbered Destination Single-Registers

The binary scaling of the product is equal to the multiplier scale factor plus the multiplicand scale factor.

Appropriate Condition Codes are set if the product is zero or negative, or if it contains less than 32 or 16 significant bits, respectively, for double-register or single-register operations. Overflow will not occur and always is cleared.

MPR,r,s — Multiply Register By Register

0 7	8 11	12 15
20	Rd	Rs

(Rs)x(RdV1) → (Rd:RdV1)

The contents of register Rs (multiplicand) are multiplied by the contents of register RdV1 (multiplier). The product replaces the contents of register Rd:RdV1. The sign of the product replaces the sign bit of register Rd.

Affected: (Rd:RdV1)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set if result <0, otherwise cleared
- C Set if less than 16 significant bits in result, otherwise cleared
- O Cleared

MPRD,r,s — Multiply Double-Register By Double-Register

0 7	8 11	12		15
A2	Rd	0	Rs	0

(Rs:RsV1)x(RdV2:RdV3) → (Rd:RdV1:RdV2:RdV3)

The contents of registers Rs:RsV1 (multiplicand) are algebraically multiplied by the contents of registers RdV2: RdV3 (multiplier). The product replaces the contents of registers Rd:RdV1:RdV2:RdV3. The sign of the product replaces the sign bit of register Rd.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set if result < 0, otherwise cleared
- C Set if less than 32 significant bits in result
- O Cleared

MPM*,r,x A — Multiply Register By Memory

0 7	8 11	12	15
A0	Rd	I	Rxx
Virtual Operand Address			

(VEA)x(RdV1) → (Rd:RdV1)

The contents of memory location VEA (multiplicand) are multiplied by the contents of register RdV1 (multiplier). The product replaces the contents of registers Rd:RdV1. The sign of the product replaces the sign bit of register Rd.

Affected: (Rd:RdV1)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set if result < 0, otherwise cleared
- C Set if less than 16 significant bits in result, otherwise cleared
- O Cleared

MPI,r V — Multiply Immediate

0 7	8 11	12 15
E8	Rd	6
Immediate Operand		

(RdV1)x((PR)+1) → (Rd:RdV1)

The contents of register RdV1 are multiplied by the contents of the immediate memory location. The result is placed in Rd:RdV1.

Affected: (Rd:RdV1)

Condition Code:

- Z Set if result = 0, otherwise cleared
- N Set if result < 0, otherwise cleared
- C Set if less than 16 significant bits in result, otherwise cleared
- O Cleared

MPS,r,d — Multiply Register By Memory (Short-Displaced)

0 7	8 11	12 15
B0	Rd	D

((R1)+D)x(RdV1) ⟶(Rd:RdV1)

The contents of short-displaced memory location VEA (multiplicand) are multiplied by the contents of register RdV1 (multiplier). The product replaces the contents of register Rd:RdV1. The sign of the product replaces the sign bit of register Rd.

Affected: (Rd:RdV1)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set if result < 0, otherwise cleared
- C Set if less than 16 significant bits in result, otherwise cleared
- O Cleared

MPX,r,x — Multiply Register By Memory (Short-Indexed)

0 7	8 11	12 15
B8	Rd	Rx

((Rx))x(RdV1) ⟶(Rd:RdV1)

The contents of short-indexed memory location VEA (multiplicand) are multiplied by the contents of register RdV1 (multiplier). The product replaces the contents of register Rd:RdV1. The sign of the product replaces the sign bit of register Rd.

Affected: (Rd:RdV1)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set if result < 0, otherwise cleared
- C Set if less than 16 significant bits in result, otherwise cleared
- O Cleared

MPMD,r,x A — Multiply Double-Register By Memory Doubleword

0 7	8	10	11	12 15
A2	Rd	0 1	I	Rxx
Virtual Operand Address				

(VEA:VEA+1)x(RdV2:RdV3) ⟶(Rd:RdV1:RdV2:RdV3)

The contents of memory location VEA:VEA+1 (multiplicand) are algebraically multiplied by the contents of registers Rd:RdV1 (multiplier). The product replaces the contents of registers Rd:RdV1:RdV2:RdV3. The sign of the product replaces the sign bit of register Rd.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set if result < 0, otherwise cleared
- C Set if less than 32 significant bits in result
- O Cleared

Divide Register(s) by Operand

The following instructions divide fixed point word or double-word operands contained in the general purpose registers by operands in registers or memory.

The format and positioning of operands for divide operations are compatible with and are described with the preceding multiply operations (see figures 9-4 through 9-6).

During execution of the divide instructions, the dividend contained in registers Rd:RdV1:RdV2:RdV3 or Rd:RdV1 is divided by the corresponding double-word or single-word divisor specified by the instruction. If the dividend is smaller than the register it occupies, it should be sign extended prior to instruction execution or erroneous results may occur. For double-word divides Rd must be R4, R8 or R12. For single-word divides Rd must be even.

After execution of the divide, the resulting double-word or single-word quotient normally replaces the low order contents of quad- or double-register Rd and the remainder replaces the high order contents of quad- or double-register Rd.

The sign of the quotient is positive if the dividend and divisor have like signs; otherwise it is negative. If the magnitude of the quotient is so small as to require more than 31 (double-register) or 15 (single-register) bits to resolve, a zero quotient may result. The binary scaling of the quotient is equal to the dividend scale factor minus the divisor scale factor.

The remainder has the same sign as the dividend. It has the same scaling as the divisor. By definition, the undivided remainder is the remaining portion of the dividend that has not been divided. Thus, it may be added to the least significant part of the product of the divisor and the quotient to reproduce the original dividend with the proper scaling.

If the same double- or quad-register is specified for both source and destination, the remainder and quotient replace the original operand and are +0 and +1 respectively.

Overflow will occur if the quotient contains more than 32 or 16 significant bits, respectively, for double-register or single-register operations. This condition results when the sign and most significant bit of the dividend are not equal, or when the absolute magnitude of the most significant half of the dividend (shifted left one bit position) is not less than the absolute magnitude of the divisor.

Appropriate Condition Codes are set if the quotient is zero or negative, or if it overflows. Carry will be set if the divisor is zero.

DVR,r,s — Divide Register By Register

0 – 7	8 – 11	12 – 15
21	Rd	Rs

(Rd:RdV1)/(Rs) ⟶ (Rd:RdV1) REM:QNT

The contents of registers Rd:RdV1 (dividend) are divided by the contents of register Rs (Divisor). The quotient replaces the contents of register RdV1 and the remainder replaces the contents of register Rd. The sign of the quotient replaces the sign bit of register RdV1. Rd (dividend) must specify register pair.

Affected: (Rd:RdV1)

Condition Codes:
- Z Set if quotient = 0, otherwise cleared
- N Set if quotient < 0, otherwise cleared
- C Set if divisor = 0, otherwise cleared
- O Set if quotient exceeds 16 bits, otherwise cleared

DVRD,r,s — Divide Quad-Register By Double-Register

0 – 7	8 – 10	11	12 – 14	15
A3	Rd	0	Rs	0

(Rd:RdV1:RdV2:RdV3)/(Rs:RsV1) ⟶ (Rd:RdV1:RdV2:RdV3) REM:QNT

The contents of registers Rd:RdV1:RdV2:RdV3 (dividend) are divided by the contents of registers Rs:RsV1 (divisor). The quotient replaces the contents of registers RdV2:RdV3 and the remainder replaces the contents of registers Rd:RdV1;

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:
- Z Set if quotient = 0, otherwise cleared
- N Set if quotient < 0, otherwise cleared
- C Set if divisor = 0, otherwise cleared
- O Set if the quotient exceeds 32 bits, or if bit 0 of RdV2 is different than the algebraic sign of the quotient

DVM*,r,x A — Divide Register by Memory

0 7	8 11	12	13 15
A1	Rd	I	Rx
Virtual Operand Address			

(Rd:RdV1) / (VEA) → (Rd:RdV1) REM:QNT

The contents of registers Rd:RdV1 (dividend) are divided by the contents of memory location VEA (divisor). The quotient replaces the contents of register RdV1 and the remainder replaces the contents of register Rd. The sign of the quotient replaces the sign bit of register RdV1. Rd (dividend) must specify register pair.

Affected: (Rd:RdV1)

Condition Codes;
Z Set if quotient = 0, otherwise cleared
N Set if quotient < 0, otherwise cleared
C Set if divisor = 0, otherwise cleared
O Set if quotient exceeds 16 bits, otherwise cleared

DVI,r V — Divide Immediate

0 7	8 11	12 15
E8	Rd	7
Immediate Operand		

(Rd:RdV1) / ((PR)+1) → (Rd:RdV1) REM:QNT

The contents of register Rd:RdV1 are divided by the contents of the immediate memory location. The quotient replaces the contents of register RdV1, and the remainder replaces the contents of Rd. Rd must specify an even numbered register. Rd must specify a register pair.

Affected: Rd:RdV1

Condition Codes:
Z Set if quotient = 0, otherwise cleared
N Set if quotient < 0, otherwise cleared
C Set if divisor = 0, otherwise cleared
O Set if quotient exceeds 16 bits, otherwise cleared

DVS,r,d — Divide Register by Memory (Short-Displaced)

0 7	8 11	12 15
B1	Rd	D

(Rd:RdV1) / ((R1)+D) → (Rd:RdV1) REM:QNT

The contents of registers Rd:RdV1 (dividend) are divided by the contents of short-displaced memory location VEA (divisor). The quotient replaces the contents of register RdV1 and the remainder replaces the contents of register Rd. The sign of the quotient replaces the sign bit of register RdV1. Rd must specify a register pair.

Affected (Rd:RdV1)

Condition Codes:
Z Set if quotient = 0, otherwise cleared
N Set if quotient < 0, otherwise cleared
C Set if divisor = 0, otherwise cleared
O Set if quotient exceeds 16 bits, otherwise cleared

DVX,r,x — Divide Register by Memory (Short-Indexed)

0 7	8 11	12 15
B9	Rd	Rx

(Rd:RdV1) / ((Rx)) → (Rd:RdV1) REM:QNT

The contents of registers Rd:RdV1 (dividend) are divided by the contents of short-indexed memory location VEA (divisor). The quotient replaces the contents of register RdV1 and the remainder replaces the contents of register Rd. The sign of the quotient replaces the sign bit of register RdV1. Rd must specify register pair.

Affected: (Rd:RdV1)

Condition Codes:
Z Set if quotient = 0, otherwise cleared
N Set if quotient < 0, otherwise cleared
C Set if divisor = 0, otherwise cleared
O Set if quotient exceeds 16 bits, otherwise cleared

DVMD

DVMD,r,x A — Divide Quad-Register By Memory Doubleword

0 7	8		11	12	15
A3	Rd	0	1	I	Rxx
Virtual Operand Address					

(Rd:RdV1:RdV2:RdV3) / (VEA:VEA+1)→(Rd:RdV1:RdV2:RdV3) REM:QNT

The contents of registers Rd:RdV1:RdV2:RdV3 (dividend) are algebraically divided by memory locations VEA:VEA+1 (divisor). The quotient replaces the contents of registers RdV2:RdV3 and the remainder replaces the contents of registers Rd:RdV1.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:

- Z Set if quotient = 0, otherwise cleared
- N Set if quotient < 0, otherwise cleared
- C Set if divisor = 0, otherwise cleared
- O Set if the quotient exceeds 32 bits or if bit 0 of RdV2 is different than the algebraic sign of the quotient.

Transfer Two's Complement of Register(s) to Register(s) and Branch if Nonzero

The following instructions transfer the two's complement of fixed-point (or floating-point) word, double-word or quadruple-word operands contained in the general purpose registers. The source operand is not changed, unless the source register(s) and destination register(s) are specified to be the same. The single word operation may be specified to branch if the result is not zero.

Overflow will occur if the source operand is minus full scale (i.e., -2^{15}, 2^{47}, -2^{31} or -2^{63} for respective word, double-word, triple-word and quadruple-word operands).

Appropriate Condition Codes are set if the result is zero, negative, or to indicate adder carry out or overflow.

TTR,r,s — Transfer Two's Complement of Register to Register

0 7	8 11	12 15
6F	Rd	Rs

$(\overline{Rs})+1 \longrightarrow (Rd)$

The contents of register Rd are replaced by the two's complement of register Rs.

Affected: (Rd)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Equal to bit 0 of result
- C Set if carry out of most significant bit, otherwise cleared
- O Set if operand is minus full scale, otherwise cleared

TTRB,r,s B — Transfer Two's Complement of Register to Register and Branch if Nonzero

0 7	8 11	12 15
7F	Rd	Rs
Virtual Branch Address		

$(\overline{Rs})+1 \longrightarrow (Rd)$
If result does not =0,((PR)+1) → (PR)
If result =0, (PR)+2 → (PR)

The contents of register Rd are replaced by the two's complement of the contents of register Rs. If the result does not equal zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: (Rd)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Equal to bit 0 of result
- C Set if carry out of most significant bit, otherwise cleared
- O Set if operand is minus full scale, otherwise cleared cleared

TTRD,r,s — Transfer Two's Complement of Double-Register to Double-Register

0 7	8	11	12	15
8A	Rd	0	Rs	0

$(\overline{Rs:RsV1}:+1 \longrightarrow (Rd:RdV1)$

The two's complemented contents of register Rs:RsV1 are transferred to regsiters Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Equal to bit 0 of result
- C Set if carry out of most significant bit, otherwise cleared
- O Set if operand is minus full scale, otherwise cleared

TTRQ,r,s — Transfer Two's Complement of Quad-Register to Quad-Register

0 7	8		11	12		15
8B	Rd	0	0	Rs	0	0

$(\overline{Rs:RsV1:RsV2:RsV3})+1 \longrightarrow (Rd:RdV1:RdV2:RdV3)$

The two's complemented contents of registers Rs:RsV1:RsV2:RsV3 are transferred to registers Rd:RdV1:RdV2:RdV3.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Equal to bit 0 of result
- C Set if carry out of most significant bit, otherwise cleared
- O Set if operand is minus full scale, otherwise cleared

Extend Sign of Register(s)

The following instructions extend the sign of fixed-point word and double-word operands contained in the general purpose registers and store the equivalent 32-bit or 64-bit result back into the registers. The source operand is not changed unless the source register(s) and destination register(s) are specified to be the same.

Appropriate Condition Codes are set if the result is zero or negative. Overflow will not occur and always is cleared.

ESS,r,s Extend Sign Single

0 7	8	11	12 15
8C	Rd	1	Rs

$(Rs)_0 \longrightarrow (Rd)_{0-15}$
$(Rs) \longrightarrow (RdV1)$

The contents of register RdV1 are replaced by the contents of register Rs. Then, the sign bit of register Rs is propagated into all bit positions of register Rd.

Affected: (Rd:RdV1)

Condition Codes:

Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Cleared
O Cleared

ESD,r,s Extend Sign Double

0 7	8		11	12	15
8C	Rd	0	0	Rs	0

$(Rs)_0 \longrightarrow (Rd{:}RdV1)_{0-31}$
$(Rs{:}RsV1) \longrightarrow (RdV2{:}RdV3)$

The contents of registers RdV2:RdV3 are replaced by the contents of registers Rs:RsV1. Then, the sign bit of register Rs is propaged into all bit positions of register Rd:RdV1.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:

Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Cleared
O Cleared

Extend Sign Immediate and Operate

The following instructions utilize the contents of the immediate memory location, which is extended into a 32-bit operand. This operand is then used to perform the indicated operation with registers Rd:RdV1. The result is then stored in Rd:RdV1. Appropriate condition codes will be set.

LDES,r V — Load Immediate and Extend Sign

0 7	8 11	12	15
E8	Rd	0	1
Immediate Operand			

$((PR)+1) \rightarrow (RdV1)$

$RdV1_0 \rightarrow Rd_{0\text{-}15}$

The contents of the immediate memory location replace the contents of register RdV1. Then the sign bit of the immediate memory location is propagated into all bit positions of register Rd.

Affected: (Rd:RdV1)

Condition Codes:

- Z Set if all registers loaded = 0, otherwise cleared
- N Set if bit 0 of Rd = 1, otherwise cleared
- C Unaffected
- O Unaffected

ADES,r V — Add Immediate With Extended Sign

0 7	8 11	12	15
E8	Rd	0	2
Immediate Operand			

$((PR)+1)_0 \rightarrow (A)_{0\text{-}15}$

$(A):((PR)+1)+(Rd:RdV1) \rightarrow (Rd:RdV1$

The sign bit of the immediate memory location is extended 16 bits to form a 32-bit operand, and this 32-bit operand is algebraically added to the contents of register Rd:RdV1. The 32-bit result replaces the contents of register Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set if result < 0, otherwise cleared
- C Set if carry out of most significant bit, otherwise cleared
- O Set if both operands were of same sign and the result was of the opposite sign, otherwise cleared

SUES,r V — Subtract Immediate With Extended Sign

0 7	8	11	12 15
E8	Rd	0	3
Immediate Operand			

$((PR)+1)_0 \rightarrow (A)$

$(Rd:RdV1) - (A):((PR)+1 \rightarrow (Rd:RdV1)$

The sign bit of the immediate memory location is extended 16 bits to form a 32-bit operand and this 32-bit operand is algebraically subtracted from the contents of register Rd:RdV1. The 32-bit result replaces the contents of register Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result, otherwise cleared
- C Set if carry out of most significant bit
- O Set if operands were of opposite signs and the sign of the subtrahend was the same as the sign of the result, otherwise cleared

MPES,r V — Multiply Immediate With Extended Sign

0 7	8 10	11	12 15
0E	Rd	0	#A
Immediate Operand			

$((PR)+1)_0 \rightarrow (A)_{0-15}$

$(A):((PR)+1)x(RdV2:RdV3) \rightarrow (Rd:RdV1:RdV2:RdV3)$

The contents of the immediate memory location are sign extended to form a 32-bit multiplicand which is then multiplied by the contents of the concatenated register pair RdV2:RdV3. The result is placed in the concatenated quad register Rd:RdV1:RdV2:RdV3.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes
Codes:
- Z Set if result = 0, otherwise cleared
- N Set if result < 0, otherwise cleared
- C Set if less than 16 significant bits in result, otherwise cleared
- O Cleared

DVES,r V — Divide Immediate With Extended Sign

0 7	8 9	10	11	12 15
0E	Rd	0	0	#B
Immediate Operand				

$((PR)+1)_0 \rightarrow (A)_{0-15}$

$(Rd:RdV1:RdV2:RdV3(/ (A):((PR)+1) \rightarrow (Rd:RdV1:RdV2:$
$\rightarrow RdV3)$ REM:QNT

The contents of the immediate memory location are sign extended to form a 32-bit divisor which is then divided into the contents of the concatenated quad register Rd:RdV1:RdV2:RdV3. The quotient is placed in registers Rd:RdV1.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:
- Z Set if quotient = 0, otherwise cleared
- N Set if quotient < 0, otherwise cleared
- C Set if divisor = 0, otherwise cleared
- O Set if quotient exceeds 16 bits, otherwise cleared

CIES,r V — Compare Immediate With Extended Sign

0 7	8	11	12 15
E8	Rd	0	4
Immediate Operand			

$((PR)+1_0 \rightarrow (A)$

$(Rd:RdV1) - (A):((PR)+1)$ Result

The sign bit of the contents of the immediate memory location is extended to form a 32-bit operand, and this operand is algebraically subtracted from the contents of register Rd:RdV1. The result is not loaded.

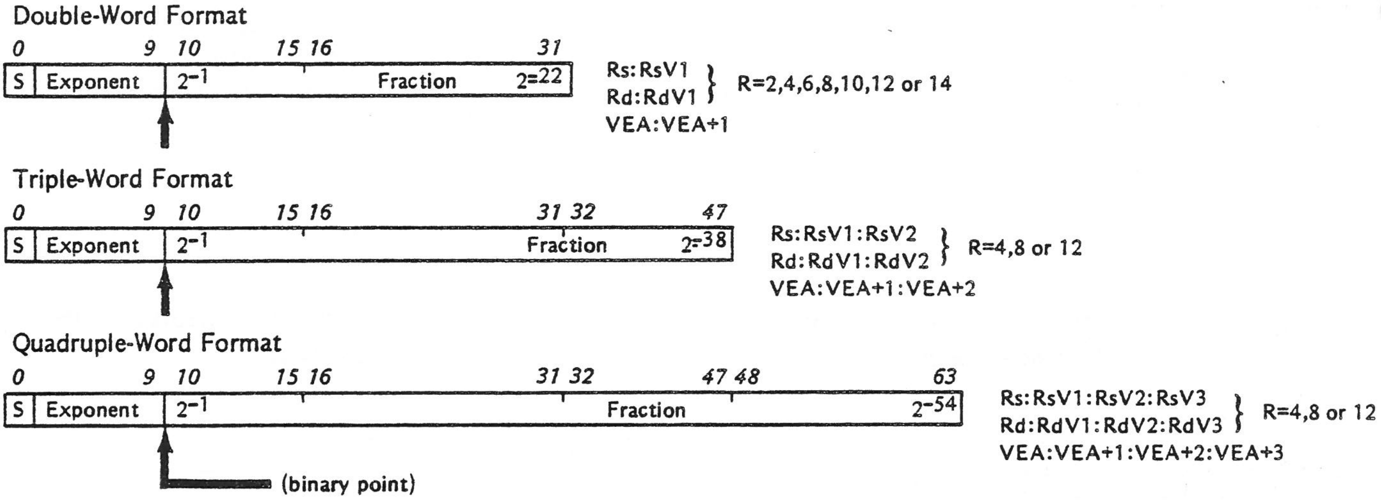
Affected: none

Condition Codes:
- Z Set if operands are equal (result = 0), otherwise cleared
- N Set if bit 0 of Result = 1, otherwise cleared
- C Set if carry from most significant bit of result, otherwise cleared
- O Set if operands were of opposite signs and the sign of the second operand is the same as the result, otherwise cleared

Floating-Point Arithmetic Instructions

This instruction class provides the capability to perform add, subtract, multiply, divide and convert operations on multi-word floating-point operands. Double-word, triple-word and quadruple-word operations may take place in the general purpose registers, between the registers and memory and between registers and immediate operands. The memory operations use only direct and indexed (R1 through R15) addressing modes.

All of the instructions in this class set Condition Codes. All operations assume normalized floating point fraction-with-exponent operands. All results are normalized and fractionally scaled by hardware. These floating-point operations also assume that operand register positioning and unit length are those designated by the operation code, and that operand formats are consistent with the following standard specification for floating-point numbers.



Double-Word Format

Triple-Word Format

Quadruple-Word Format

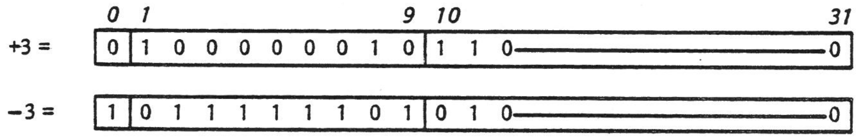
Only even numbered registers may be specified for double-word operands, not including R0. R4, R8, R12 must be specified for larger operands, otherwise the results are invalid.

Floating-point numbers consist of three parts: a sign, an exponent, and a fraction. The sign bit applies only to the fraction. The exponent is a biased nine-bit binary number. The fraction is a binary number with an assumed radix point to the left of the high-order bit. The quantity that the floating point number represents is obtained by multiplying the fraction value by 2 to the power expressed in the exponent value.

If $Q = Fx2^E$ (where Q is the quantity represented and $1/2 \leqslant F < 1$) the exponent equals E plus a bias of 200_{16}.

The first bit (bit 0) in all formats is the sign of the fraction. A one (1) bit represents a minus sign and a zero bit (0) represents a positive sign. The next nine bits (bits 1-9) represent the biased binary exponent. The fraction starts in bit position 10 and contains either a 22-bit binary number (double-word format) a 38-bit binary number (triple-word format), or a 54-bit binary number (quadruple-word format) in two's complement representation.

A negative floating-point number is expressed as the two's complement of the positive equivalent. This results in S equal to 1, the exponent equal to the one's complement of the positive exponent and the fraction equal to the two's complement of the positive fraction. For example:



Floating-point operands should be prepared in normalized form; the instructions always produce normalized results. An unnormalized number may be normalized by adding it to zero. A floating point zero should be represented by a pure binary zero, by zeroing all three parts of the number.

Floating-point overflow/underflow occurs if the resultant exponent of a floating point operation cannot be expressed within the range of the nine-bit exponent field of the floating point format. In division operations, overflow/underflow also will occur if the divisor is equal to zero.

A special form of overflow/underflow is detected if the conversion of a double-word floating-point number to double-word fixed point form results in an integer too large to be represented in a signed 32-bit quantity.

A trap occurs at interrupt level 5 if floating-point overflow/underflow is detected. (See Chapter 6, Priority Interrupts, for a detailed explanation of traps.)

All floating-point instructions set appropriate Condition Codes if the result is zero, or to indicate overflow/underflow. Precision loss in fixed point and floating point conversion operations is detected and indicated as a special form of carry.

Floating Point Load Immediate to Registers

The following instructions load the contents of immediate memory location VEA into the selected destination register. In all cases, the unloaded portions of the double, triple or quadruple-register sets are reset to zero.

LDF,r V — Load Immediate Floating Point

0 7	8 11	12 15
ED	Rd	1
Immediate Operand		

(PR+1) → (Rd)
0 → (RdV1)

The contents of the immediate memory location are loaded into register Rd. Zeros are loaded into all bit positions of register RdV1.

Affected: (Rd:RdV1)

Condition Codes:
Z Set if all registers loaded = 0, otherwise cleared
N Set if bit 0 of Rd = 1, otherwise cleared
C Unaffected
O Unaffected

LDFD,r V — Load Immediate Floating Point Double Precision

0 7	8 11	12 15
ED	Rd	2
Immediate Operand		

((PR)+1) → (Rd)
0 → (RdV1:RdV2)

The contents of the immediate memory location are loaded into register Rd. Zeros are loaded into all bit positions of register RdV1:RdV2.

Affected: (Rd:RdV1:RdV2)

Condition Codes:
Z Set if all registers loaded = 0, otherwise
N cleared
N Set if bit 0 of Rd = 1, otherwise cleared
C Unaffected
O Unaffected

LDFQ,r V — Load Immediate Floating Point Quad Precision

0 7	8 11	12 15
ED	Rd	3
Immediate Operand		

((PR)+1) → (Rd)
0 → (RdV1:RdV2:RdV3)

The contents of the immediate memory location are loaded into register Rd. Zeros are loaded into all bit positions of register RdV1:RdV2:RdV3.

Condition Codes:
Z Set if all registers loaded = 0, otherwise cleared
N Set if bit 0 of Rd = 1, otherwise cleared
C Unaffected
O Unaffected

Floating Point Add Operand to Registers

The following instructions add floating-point double-word, triple-word and quadruple-word operands contained in registers or memory to the general purpose registers.

FAR,r,s Floating Point Add Double-Register to Double-Register

0 7 8 11 12 15

30	Rd	0	Rs	0

(Rs:RsV1) + (Rd:RdV1) → (Rd:RdV1)

The contents of registers Rs:RsV1 (augend) are algebraically added to the contents of registers Rd:RdV1 (addend). The sum replaces the contents of registers Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:

Z Set if result = 0, otherwise cleared

N Set equal to bit 0 of result

C Reset if result overflows, set if result underflows

O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

FARD,r,s Floating Point Add Triple-Register to Triple-Register

0 7 8 11 12 15

34	Rd	0	0	Rs	0	0

(Rs:RsV1:RsV2) + (Rd:RdV1:RdV2) → (Rd:RdV1:RdV2)

The contents of registers Rs:RsV1:RsV2 (augend) are algebraically added to the contents of registers Rd:RdV1:RdV2 (addend). The sum replaces the contents of registers Rd:RdV1:RdV2.

Affected: (Rd:RdV1:RdV2)

Condition Codes:

Z Set if result = 0, otherwise cleared

N Set equal to bit 0 of result

C Reset if result overflows, set if result underflows

O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

FARQ,r,s — Floating Point Add Quad-Register to Quad-Register

0 7 8 11 12 15

34	Rd	0	1	Rs	0	0

(Rs:RsV1:RsV2:RsV3) + (Rd:RdV1:RdV2:RdV3)→ (Rd:RdV1:RdV2:RdV3)

The contents of registers Rs:RsV1:RsV2:RsV3 (augend) are algebraically added to the contents of registers Rd:RdV1:RdV2:RdV3 (addend). The sum replaces the contents of registers Rd:RdV1:RdV2:RdV3.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:

Z Set if result = 0, otherwise cleared

N Set equal to bit 0 of result

C Reset if result overflows, set if result underflows

O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

FAM,r,x A — Floating Point Add Memory Doubleword to Double-Register

0 7 8 11 12 15

38	Rd	0	Rxxx
Virtual Operand Address			

(VEA:VEA+1) + (Rd:RdV1) →(Rd:RdV1)

The contents of memory locations VEA:VEA+1 (augend) are algebraically added to the contents of registers Rd:RdV1 (addend). The sum replaces the contents of registers Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:

Z Set if result = 0, otherwise cleared

N Set equal to bit 0 of result

C Reset if result overflows, set if result underflows

O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

FAI,r V Floating Point Add Immediate

0 7 8 11 12 15

38	Rd	1	0
Immediate Operand			

((PR)+1):0+(Rd:RdV1) →(Rd:RdV1)

The contents of the immediate memory location (augend) are treated as the most significant 16 bits of a floating point number with least significant bits of zero. The resultant floating point number is algebraically added to the contents of registers Rd:RdV1 (addend). The sum replaces the contents of registers Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Reset if result overflows, set if result underflows
- O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

FAMD,r,x A Floating Point Add Memory Tripleword to Triple-Register

0 7 8 11 12 15

3C	Rd	0	0	Rxxx
Virtual Operand Address				

(VEA:VEA+1:VEA+2) + (Rd:RdV1:RdV2) → (Rd:RdV1:RdV2)

The contents of memory locations VEA:VEA+1:VEA+2 (augend) are algebraically added to the contents of registers Rd:RdV1:RdV2 (addend). The sum replaces the contents of registers Rd:RdV1:RdV2.

Affected: (Rd:RdV1:RdV2)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Reset if result overflows, set if result underflows
- O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

FAID,r V Floating Point Add Immediate Double Precision

0 7 8 11 12 15

3C	Rd	1	0	0
Immediate Operand				

((PR)+1)):0:0+(Rd:RdV1:RdV2) ⟶(Rd:RdV1:RdV2)

The contents of the immediate memory location (augend) are treated as the most significant 16 bits of a floating-point number with least significant bits of zero. The resultant floating-point number is algebraically added to the contents of registers Rd:RdV1:RdV2 (addend). The sum replaces the content of registers Rd:RdV1:RdV2.

Affected: (Rd:RdV1:RdV2)

Condition Codes:

Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Reset if result overflows, set if result underflows
O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

FAMQ,r,x A Floating Point Add Memory Quadword to Quad-Register

0 7 8 11 12 15

3C	Rd	0	1	Rxxx
Virtual Operand Address				

(VEA:VEA+1:VEA+2:VEA+3)+(Rd:RdV1:RdV2:RdV3)⟶
(Rd:RdV1:RdV2:RdV3)

The contents of memory locations VEA:VEA+1:VEA+2:VEA+3 (augend) are algebraically added to the contents of registers Rd:RdV1:RdV2:RdV3 (addend). The sum replaces the contents of registers Rd:RdV1:RdV2:RdV3.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:

Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Reset if result overflows, set if result underflows
O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

FAIQ,r V Floating Point Add Immediate Quad Precision

0 7	8		11 12	15
3C	Rd	1	1	0
Immediate Operand				

((PR)+1):0:0:0+(Rd:RdV1:RdV2:RdV3) → (Rd:RdV1:RdV2:RdV3)

The contents of the immediate memory location (augend) are treated as the most significant 16 bits of a floating point number with least significant bits of zero. The resultant floating point number is algebraically added to the contents of registers Rd:RdV1:RdV2:RdV3 (addend). The sum replaces the contents of registers Rd:RdV1:RdV2:RdV3.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Reset if result overflows, set if result underflows
- O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared.

Floating Point Subtract Operand from Registers

The following instructions subtract floating-point doubleword, triple-word, and quadruple-word operands contained in registers or memory from the general purpose registers.

FSR,r,s — Floating Point Subtract Double-Register From Double-Register

0 7	8	11 12		15
31	Rd	0	Rs	0

(Rd:RdV1)-(Rs:RsV1) ⟶(Rd:RdV1)

The contents of registers Rs:RsV1 (subtrahend) are algebraically subtracted from the contents of registers Rd:RdV1 (minuend). The difference is stored in registers Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:

Z Set if result = 0, otherwise cleared

N Set equal to bit 0 of result

C Reset if result overflows, set if result underflows

O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no OVERFLOW or UNDERFLOW, condition code 0 is cleared

FSRD,r,d — Floating Point Subtract Triple-Register From Triple-Register

0 7	8		11 12			15
35	Rd	0	0	Rs	0	0

(Rd:RdV1:RdV2)-(Rs:RsV1:RsV2) ⟶(Rd:RdV1:RdV2)

The contents of registers Rs:RsV1:RsV2 (subtrahend) are algebraically subtracted from the contents of registers Rd:RdV1;RdV2 (minuend). The difference is stored in registers Rd:RdV1:RdV2.

Affected: (Rd:RdV1:RdV2)

Condition Codes:

Z Set if result = 0, otherwise cleared

N Set equal to bit 0 of result

C Reset if result overflows, set if result underflows

O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no OVERFLOW or UNDERFLOW, condition code 0 is cleared

FSRQ,r,s — Floating Point Subtract Quad-Register From Quad-Register

0 7	8		11	12		15
35	Rd	0	T	Rs	0	0

(Rd:RdV1:RdV2:RdV3)-(Rs:RsV1:RsV2:RsV3) →(Rd:RdV1:RdV2:RdV3)

The contents of registers Rs:RsV1:RsV2:RsV3 (subtrahend) are algebraically subtracted from the contents of registers Rd:RdV1:RdV2:RdV3 (minuend). The difference is stored in registers Rd:RdV1:RdV2:RdV3.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Reset if result overflows, set if result underflows
- O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no OVERFLOW or UNDERFLOW, condition code 0 is cleared

FSM,r,x A — Floating Point Subtract Memory Doubleword From Double-Register

0 7	8 11	12 15
39	Rd	Rxxx
Virtual Operand Address		

(Rd:RdV1)-(VEA:VEA+1) →(Rd:RdV1)

The contents of memory location VEA:VEA+1 (subtrahend) are algebraically subtracted from the contents of registers Rd:RdV1 (minuend). The difference replaces the contents of registers Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Reset if result overflows, set if result underflows
- O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no OVERFLOW or UNDERFLOW, condition code 0 is cleared

FSI,r V Floating Point Subtract Immediate

0 7	8 11	12	15
39	Rd	1	0
Immediate Operand			

(Rd:RdV1)-((PR)+1):0 → (Rd:RdV1)

The contents of the immediate memory location (subtrahend) are treated as the most significant 16 bits of a floating point number with least significant bits of zero. The resultant floating point number is algebraically subtracted from the contents of registers Rd:RdV1 (minuend). The difference is stored in registers Rd:RdV1.

Affected: (Rd:RdV1)

Condtion Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Reset if result overflows, set if result underflows
- O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no OVERFLOW or UNDERFLOW, condition code 0 is cleared

FSMD,r,x A Floating Point Subtract Memory Tripleword From Triple-Register

0 7	8	11	12	15
3D	Rd	0	0	Rxxx
Virtual Operand Address				

(Rd:RdV1:RdV2)-(VEA:VEA+1VEA+2) → (Rd:RdV1:RdV2)

The contents of memory locations VEA:VEA+1:VEA+2 (subtrahend) are algebraically subtracted from registers Rd:RdV1:RdV2 (minuend). The difference replaces the contents of registers Rd:RdV1:RdV2.

Affected:(Rd:RdV1:RdV2)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Reset if result overflows, set if result underflows
- O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no OVERFLOW or UNDERFLOW, condition code 0 is cleared

FSID,r V Floating Point Subtract Immediate Double Precision

0 7	8		11	12 15
3D	Rd	1	0	0
Immediate Operand				

2

(Rd:RdV1:RdV2)-((PR)+1):0:0→(Rd:RdV1:RdV3)

The contents of the immediate memory location (subtrahend) are treated as the most significant 16 bits of a floating-point number with least significant bits of zero. The resultant floating-point number is algebraically subtracted from the contents of registers Rd:RdV1:RdV2 (minuend). The difference is stored in registers Rd:RdV1:RdV2.

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Reset if result overflows, set if result underflows
- O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no OVERFLOW or UNDERFLOW, condition code O is cleared

FSMQ,r,x A Floating Point Subtract Memory Quadword From Quad-Register

0 7	8		11	12 15
3D	Rd	0	1	Rxxx
Virtual Operand Address				

(Rd:RdV1:RdV2:RdV3)-(VEA:VEA+1:VEA+2:VEA+3)→
(Rd:RdV1:RdV2:RdV3)

The contents of memory locations VEA:VEA+1:VEA+2:VEA+3 (subtrahend) are algebraically subtracted from registers Rd:RdV1:RdV2:RdV3. The difference replaces the contents of Rd:RdV1:RdV2:RdV3.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Reset if result overflows, set if result underflows
- O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no OVERFLOW or UNDERFLOW, condition code O is cleared

FSIQ,r V — Floating Point Subtract Immediate From Quad Precision

0 7	8	11	12	15
3D	Rd	1	1	0
Immediate Operand				

(Rd:RdV1:RdV2:RdV3)-((PR)+1):0:0:0 → (Rd:RdV1:RdV2:RdV3)

The contents of the immediate memory location (subtrahend) are algebraically subtracted from the contents of registers Rd:RdV1:RdV2:RdV3 (minuend). The difference is stored in registers Rd:RdV1:RdV2:RdV3.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Reset if result overflows, set if result underflows
- O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no OVERFLOW or UNDERFLOW, condition code 0 is cleared

Floating Point Multiply Registers by Operand

The following instructions multiply floating-point double-word, triple-word, or quadruple-word operands contained in the general purpose registers by registers or memory.

FMR,r,s — Floating Point Multiply Double-Register by Double-Register

0 7	8	11	12	15
32	Rd	0	Rs	0

(Rs:RsV1) x (Rd:RdV1) → (Rd:RdV1)

The contents of registers Rs:RsV1 (multiplicand) are multiplied by the contents of registers Rd:RdV1 (multiplier). The product is stored in registers Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Reset if result overflows, set if result underflows
- O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

FMRD,r,s — Floating Point Multiply Triple-Register by Triple-Register

0 7	8		11	12		15
36	Rd	0	0	Rs	0	0

(Rs:RsV1:RsV2)x(Rd:RdV1:RdV2) → (Rd:RdV1:RdV2)

The contents of registers Rs:RsV1:RsV2 (multiplicand) are multiplied by the contents of Rd:RdV1:RdV2 (multiplier). The product is stored in registers Rd:RdV1:RdV2.

Affected: (Rd:RdV1:RdV2)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Reset if result overflows, set if result underflows
- O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

FMRQ,r,s Floating Point Multiply Quad-Register by Quad-Register

0 7	8		11	12		15
36	Rd	0	1	Rs	0	0

(Rs:RsV1:RsV2:RsV3)x(Rd:RdV1:RdV2:RdV3)→
(Rd:RdV1:RdV2:RdV3)

The contents of registers Rs:RsV1:RsV2:RsV3 (multiplicand) are multiplied by the contents of registers Rd:RdV1:RdV2:RdV3 (multiplier). The product is stored in registers Rd:RdV1:RdV2:RdV3.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:

Z Set if result = 0, otherwise cleared

N Set equal to bit 0 of result

C Reset if result overflows, set if result underflows

O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

FMM,r,x A Floating Point Multiply Double-Register by Memory Doubleword

0 7	8	11	12 15
3A	Rd	0	Rxxx
Virtual Operand Address			

(VEA):(VEA+1)x(Rd:RdV1) →(Rd:RdV1)

The contents of memory locations VEA:VEA+1 (multiplicand) are multiplied by the contents of registers Rd:RdV1 (multiplier). The product replaces the contents of registers Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:

Z Set if result = 0, otherwise cleared

N Set equal to bit 0 of result

C Reset if result overflows, set if result underflows

O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

FMI,r V Floating Point Multiply Immediate

0 7 8 11 12 15

3A	Rd	1	0
Immediate Operand			

(Rd:RdV1) x ((PR)+1):0 → (Rd:RdV1)

The contents of the immediate memory location (multiplicand) are treated as the most significant 16 bits of a floating-point number with least significant bits of zero. The resultant floating-point number is multiplied by the contents of registers Rd:RdV1 (multiplier). The product is stored in registers Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:

Z Set if result = 0, otherwise cleared

N Set equal to bit 0 of result

C Reset if result overflows, set if result underflows

O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

FMMD,r,x A Floating Point Multiply Triple-Register by Memory Triple-Word

0 7 8 11 12 15

3E	Rd	0	0	Rxxx
Virtual Operand Address				

(VEA:VEA+1:VEA+2)x(Rd:RdV1:RdV2) → (Rd:RdV1:RdV2)

The contents of memory locations VEA:VEA+1:VEA+2 (multiplicand) are multiplied by the contents of registers Rd:RdV1:RdV2 (multiplier). The product replaces the contents of registers Rd:RdV1:RdV2.

Affected: (Rd:RdV1:RdV2)

Condition Codes:

Z Set if result = 0, otherwise cleared

N Set equal to bit 0 of result

C Reset if result overflows, set if result underflows

O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

FMID,r V Floating Point Multiply Immediate Double Precision

0 7 8 11 12 15

3E	Rd	1	0	0
Immediate Operand				

(Rd:RdV1:RdV2)x((PR)+1):0:0→(Rd:RdV1):(RdV2)

The contents of the immediate memory location (multiplicand) are treated as the most significant 16 bits of a floating point number with least significant bits of zero. The resultant floating point number is multiplied by the contents of registers Rd:RdV1:RdV2 (multiplier). The product is stored in registers Rd:RdV1:RdV2.

Affected: (Rd:RdV1:RdV2)

Condition Codes:

Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Reset if result overflows, set if result underflows
O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

FMMQ,r,x A Floating Point Multiply Quad-Register by Memory Quadword

0 7 8 11 12 15

3E	Rd	0	1	Rxxx
Virtual Operand Address				

(VEA:VEA+1:VEA+2:VEA+3)x(Rd:RdV1:RdV2:RdV3)→
(Rd:RdV1:RdV2:RdV3)

The contents of memory locations VEA:VEA+1:VEA+2:VEA+3 (multiplicand) are multiplied by the contents of registers Rd:RdV1:RdV2:RdV3 (multiplier). The product replaces the contents of registers Rd:RdV1:RdV2:RdV3.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:

Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Reset if result overflows, set if result underflows
O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

FMIQ,r V Floating Point Multiply Immediate Quad Precision

0 7	8		11	12 15
3E	Rd	1	1	0
Immediate Operand				

(Rd:RdV1:RdV2:RdV3)x(((PR)+1):0:0:0 → (Rd:RdV1:RdV2:RdV3)

The contents of the immediate memory location (multiplicand) are treated as the most significant 16 bits of a floating-point number with least significant bits of zero. The resultant floating-point number is multiplied by the contents of registers Rd:RdV1:RdV2:RdV3 (multiplier). The product is stored in registers Rd:RdV1:RdV2:RdV3.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Reset if result overflows, set if result underflows
- O Set if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

Floating Point Divide Registers by Operand

The following instructions divide floating-point double-word, triple-word, and quadruple-word operands contained in the General Purpose registers by registers or memory.

FDR,r,s — Floating Point Divide Double-Register by Double-Register

0 7	8	11	12	15
33	Rd	0	Rs	0

(Rd:RdV1) / (Rs:RsV1) → (Rd:RdV1)

The contents of registers Rd:RdV1 (dividend) are divided by the contents of registers Rs:RsV1 (divisor). The quotient replaces the contents of registers Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:

Z Set if result = 0, otherwise cleared

N Set equal to bit 0 of result

C Reset if result overflows, set if result underflows

O Set if divisor = 0, or if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

FDRD,r,s — Floating Point Divide Triple-Register by Triple-Register

0 7	8		11	12		15
37	Rd	0	0	Rs	0	0

(Rd:RdV1:RdV2) / (Rs:RsV1:RsV2) → (Rd:RdV1:RdV2)

The contents of registers Rd:RdV1:RdV2 (dividend) are divided by the contents of registers Rs:RsV1:RsV2 (divisor). The quotient replaces the contents of registers Rd:RdV1:RdV2.

Affected: (Rd:RdV1:RdV2)

Condition Codes:

Z Set if result = 0, otherwise cleared

N Set equal to bit 0 of result

C Reset if result overflows, set if result underflows

O Set if divisor = 0, or if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

FDRQ,r,s — Floating Point Divide Quad-Register by Quad-Register

0 – 7	8 – 11			12 – 15		
37	Rd	0	1	Rs	0	0

(Rd:RdV1:RdV2:RdV3) / (Rs:RsV1:RsV2:RsV3)→ (Rd:RdV1:RdV2:RdV3)

The contents of registers Rd:RdV1:RdV2:RdV3 (dividend) are divided by the contents of registers Rs:RsV1:RsV2:RsV3 (divisor). The quotient replaces the contents of registers Rd:RdV1:RdV2:RdV3.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Reset if result overflows, set if result underflows
- O Set if divisor = 0, or if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

FDM,r,x A — Floating Point Divide Double-Register by Memory Double-word

0 – 7	8 – 11		12 – 15
3B	Rd	0	Rxxx
Virtual Operand Address			

(Rd:RdV1) / (VEA:VEA+1) →(Rd:RdV1)

The contents of registers Rd:RdV1 (dividend) are divided by the contents of memory locations VEA:VEA+1 (divisor). The quotient replaces the contents of registers Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Reset if result overflows, set if result underflows
- O Set if divisor = 0, or if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

FDMD,r,x A Floating Point Divide Triple-Register by Memory Tripleword

0 7	8		11	12 15
3F	Rd	0	0	Rxxx
Virtual Operand Address				

(Rd:RdV1:RdV2) / (VEA:VEA+1VEA+2) → (Rd:RdV1:RdV2)

The contents of registers Rd:RdV1:RdV2 (dividend) are divided by the contents of memory locations VEA:VEA+1: VEA+2 (divisor). The quotient replaces the contents of registers Rd:RdV1:RdV2.

Affected: (Rd:RdV1:RdV2)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Reset if result overflows, set if result underflows
- O Set if divisor = 0, or if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

FDI,r V Floating Point Divide Immediate

0 7	8 11		12 15
3B	Rd	1	0
Immediate Operand			

(Rd:RdV1:RdV2) / ((PR)+1 → (Rd:RdV1:RdV2)

The contents of registers Rd:RdV1 (dividend) are divided by an operand formed by the immediate memory location (most significant bits) and zeroes as the least significant bits (divisor). The quotient replaces the contents of registers Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Reset if result overflows, set if result underflows
- O Set if divisor = 0, or if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared

FDID,r V — Floating Point Divide Immediate Double Precision

0 7	8 11	12		15
3F	Rd	1	0	0
Immediate Operand				

(Rd:RdV1:RdV2) / ((PR+1) → (Rd:RdV1:RdV2)

The contents of registers Rd:RdV1:RdV2 (dividend) are divided by an operand formed by the immediate memory location (most significant bits) and zeroes as the least significant bits (divisor). The quotient replaces the contents of registers Rd:RdV1:RdV2.

Affected: (Rd:RdV1:RdV2)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Reset if result overflows, set if result underflows
- O Set if divisor = 0, or if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW, condition code 0 is cleared

FDMQ,r,x A — Floating Point Divide Quad-Register by Memory Quadword

0 7	8 11	12		15
3F	Rd	0	1	Rxxx
Virtual Operand Address				

(Rd:RdV1:RdV2:RdV3) / (VEA:VEA+1:VEA+2:VEA+3) → (Rd:RdV1:RdV2:RdV3)

The contents of registers Rd:RdV1:RdV2:RdV3 (dividend) are divided by the contents of memory location VEA:VEA+1:VEA+2:VEA+3. The quotient replaces the contents of registers Rd:RdV1:RdV2:RdV3.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Reset if result overflows, set if result underflows
- O Set if divisor = 0, or if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW, condition code 0 is cleared

FDIQ,r V Floating Point Divide Immediate Quad Precision

0 7	8		11	12 15
3F	Rd	1	1	0
Immediate Operand				

(Rd: RdV1: RdV2: RdV3) / ((PR)+1) → (Rd: RdV1: RdV2: RdV3)

The contents of registers Rd: RdV1: RdV2: RdV3 (dividend) are divided by an operand formed by the immediate memory location (most significant bits) and zeroes as the least significant bits (divisor). The quotient replaces the contents of registers Rd: RdV1: RdV2: RdV3.

Affected: (Rd: RdV1: RdV2: RdV3)

Condition Codes:

Z Set if result = 0, otherwise cleared

N Set equal to bit 0 of result

C Reset if result overflows, set if result underflows

O Set if divisor = 0, or if resultant exponent is too large to be expressed within the range of 9 bits (OVERFLOW) or if the resultant number is too small to be expressed within the range of 9 bits (UNDERFLOW). Both UNDERFLOW and OVERFLOW will cause a trap at interrupt level 5. If there is no UNDERFLOW or OVERFLOW, condition code 0 is cleared.

Convert Operand in Registers

The following instructions convert fixed point operands to or from floating-point operands within the general purpose registers. Floating point double-words are converted to fixed-point double-words, fixed-point double-words are converted to floating-point double-words, and floating-point quadruple-words are converted to floating-point double-words. The magnitude of the result is less than or equal to the absolute value of the source operand.

CFDI,r,s — Convert Floating Point to Double-Register Integer

0 … 7	8 … 11	12	… 15	
34	Rd	0	Rs	1

$(Rs:RsV1)_{FLT} \longrightarrow (Rd:RdV1)_{INT}$

The floating-point value contained in registers Rs:RdV1 is converted to integer format. The result replaces the contents of registers Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Set if precision lost in conversion, otherwise cleared
- O Set if integer too large to be represented in a 32-bit quantity, otherwise cleared

CDFI,r,s — Convert Double Precision Floating Point Operand to Double Integer

0 … 7	8 … 11	12			… 15
36	Rd	0	Rs	0	1

$(Rs:RsV1:RsV2)_{FLT} \longrightarrow (Rd:RdV1)_{INT}$

The floating-point value in Rs:RsV1:RsV2 is converted to an an integer value and placed in register Rd:RdV1.

Affected: Rd:RdV1

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Set if precision lost in conversion, otherwise cleared
- O Set if integer too large to be represented in a 32-bit quantity, otherwise cleared

CDIF,r,s — Convert Double-Register Integer to Floating Point

0 … 7	8 … 11	12	… 15	
30	Rd	0	Rs	1

$(Rs:RsV1)_{INT} \longrightarrow (Rd:RdV1)_{FLT}$

The integer value contained in registers Rs:RsV1 is converted to floating point format. The result replaces the contents of registers Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Set if precision lost in conversion, otherwise cleared
- O Cleared

CQFF,r,s — Convert Quad-Register Floating Point to Floating Point

0 … 7	8 … 11	12			… 15
35	Rd	0	Rs	0	1

$(Rs:RsV1:RsV2:RsV3)_{FLT} \longrightarrow (Rd:RdV1)_{FLT}$

The extended precision floating-point value contained in registers Rs:RsV1:RsV2:RsV3 is converted to floating-point format. The result replaces the contents of registers Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Set if precision lost in conversion, otherwise cleared
- O Set if the resultant exponent is too large to be expressed within the range of nine bits

CQFI,r,s — Convert Quad-Precision Floating Point Operand to Double Integer

0 – 7	8 – 11	12 – 15			
37	Rd	0	Rs	0	1

$(Rs{:}RsV1{:}RsV2{:}RsV3)_{FLT} \rightarrow (Rd{:}RdV1)_{INT}$

The floating-point value in Rs:RsV1:RsV2:RsV3 is converted to an integer value and placed in register Rd:RdV1.

Affected: (Rd:RdV1)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Set if precision lost in conversion, otherwise cleared
- O Set if integer too large to be represented in a 32-bit quantity, otherwise cleared.

Logical Instructions

This instruction class provides the capability to perform logical product (Extract), sum (OR), modulo-two sum (Exclusive OR), Zero, and One's Complement on bit and word operands in the general purpose registers or between the registers and memory. Bit operations include logical OR, Exclusive OR and Zero in registers or memory. In addition, instructions are included to set and reset the condition code registers (N,Z,O,C) and set or reset a selected register depending on the states of N,Z,O,C.

Most of the standard addressing modes are provided for operations with memory, and extended addressing of memory operands are available on indexed instructions.

Optional program branching on non-zero result is provided with some of these instructions. All of the instructions in this class set Condition Codes.

Examples of the specific functions performed by this class of logical instructions follow:

Extract

0101	source operand
1010	one's complement of source operand
0011	original destination operand
0010	final destination operand

OR

0101	source operand
0011	original destination operand
0111	final destination operand

Exclusive OR

0101	source operand
0011	original destination operand
0110	final destination operand

One's Complement

0101	source operand
1010	destination operand

Extract Word from Register (and Branch if Nonzero)

The following instructions logically multiply (AND function) the one's complement of word operands contained in the general purpose registers by operands in registers or memory. The operation in registers may be specified to branch if the result is not zero.

Appropriate Condition Codes are set if the result is zero or negative. Carry and overflow cannot occur and their Condition Codes are unaffected.

ETR,r,s Extract Register From Register

0 7	8 11	12 15
6A	Rd	Rs

$(\overline{Rs})$ .AND. (Rd) → (Rd)

The one's complement of the contents of register Rs are logically multiplied (AND function) by the contents of register Rd. The result is stored in register Rd.

Affected: (Rd)

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of Rd
C Unaffected
O Unaffected

ETRB,r,s B Extract Register From Register and Branch if Nonzero

0 7	8 11	12 15
7A	Rd	Rs
Virtual Base Address		

$(\overline{Rs})$ .AND. (Rd) → (Rd)
If result does not = 0, ((PR)+1) → (PR)
If result = 0, (PR)+2 → (PR)

The one's complement of the contents of register Rs is logically multiplied (AND function) by the contents of register Rd. The result is stored in register Rd. If the result does not equal zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: (Rd)

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of Rd
C Unaffected
O Unaffected

ETM,r,x A Extract Memory From Register

0 7	8 11	12	15
E2	Rd	I	Rxx
Virtual Operand Address			

$(\overline{VEA})$ .AND. (Rd) → (Rd)

The one's complement of the contents of memory location VEA is logically multiplied (AND function) by the contents of register Rd. The result is stored in register Rd.

Affected: (Rd)

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of Rd
C Unaffected
O Unaffected

ETI,r V Extract Memory (Immediate) From Register

0 7	8 11	12 15
EA	Rd	0
Immediate Operand		

$(\overline{(PR)+1}$ .AND. (Rd) → (Rd)

The one's complement of the contents of immediate memory location VEA is logically multiplied (AND function) by the contents of register Rd. The result is stored in register Rd.

Affected: (Rd)

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of Rd
C Unaffected
O Unaffected

TETI,r V — Test Extract Memory (Immediate) From Register

0 – 7	8 – 11	12 – 15
EA	Rd	1
Immediate Operand		

(Rd) .AND. $\overline{((PR)+1}$ → Result

The one's complement of the contents of immediate memory location VEA are logically multiplied (AND function) by the contents of register Rd. The result is not stored.

Affected: none

Condition Codes:

Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of Rd
C Unaffected
O Unaffected

ETS,r,d — Extract Memory)Short-Displaced) From Register

0 – 7	8 – 11	12 – 15
Fs	Rd	D

$\overline{((R1)+D)}$ .AND. (Rd) →(Rd)

The one's complement of the contents of short-displaced memory location VEA is logically multiplied (AND function) by the contents of register Rd. The result is stored in register Rd.

Affected: (Rd)

Condition Codes:

Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of Rd
C Unaffected
O Unaffected

ETX,r,x — Extract Memory (Short-Indexed) From Register

0 – 7	8 – 11	12 – 15
FA	Rd	Rx

$\overline{((Rx))}$ .AND. (Rd) →(Rd)

The one's complement of the contents of short-indexed memory location VEA is logically multiplied (AND function) by the contents of register Rd. The result is stored in register Rd.

Affected: (Rd)

Condition Codes:

Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of Rd
C Unaffected
O Unaffected

Extract Word from Memory (and Branch if Nonzero)

The following instructions logically multiply (AND function) word operands contained in memory by the one's complement of operands in the general purpose registers. All operations may be specified to branch if the result is not zero.

Appropriate Condition codes are set if the result is zero or negative. Carry and overflow cannot occur and their Condition Codes are unaffected.

ETMM*,s,x A   Extract Register From Memory

0 7	8 11	12	15
C1	Rs	I	Rxx
Virtual Operand Address			

$(\overline{Rs})$ .AND. (VEA) → (VEA)

The one's complement of the contents of register Rs is logically multiplied (AND function) by the contents of memory location VEA. The result is stored in memory location VEA.

Affected: (VEA)

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Unaffected
O Unaffected

ETMB*,s,x A,B   Extract Register From Memory and Branch if Nonzero

0 7	8 11	12	15
C5	Rs	I	Rxx
Virtual Operand Address			
Virtual Branch Address			

$(\overline{Rs})$ .AND. (VEA) → (VEA)
If Result does not = 0,((PR)+2) → (PR)
If Result =0,(PR)+3 → (PR)

The one's complement of the contents of register Rs is logically multiplied (AND function) by the contents of memory location VEA. The result is stored in memory location VEA. If the result does not equal zero, a branch is executed to the virtual branch address; otherwise the next instruction in sequence is executed.

Affected: (VEA)

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Unaffected
O Unaffected

ETSM,s,d — Extract Register From Memory (Short-Displaced)

0 7	8 11	12 15
DI	Rs	D

$(\overline{\text{Rs}})$ .AND. ((R1)+D) → ((R1)+D)

The one's complement of the contents of register Rs is logically multiplied (AND function) by the contents of short-displaced memory location VEA. The result is stored in memory location VEA.

Affected: (VEA)

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Unaffected
O Unaffected

ETSB,s,d B — Extract Register From Memory (Short-Displaced) and Branch if Nonzero

0 7	8 11	12 15
D5	Rs	D
Virtual Branch Address		

$(\overline{\text{Rs}})$ .AND. ((R1)+D) → ((R2)+D)
If Result does not = 0, (((PR)+1) → (PR)
If Result = 0, (PR)+2 → (PR)

The one's complement of the contents of register Rs is logically multiplied (AND function) by the contents of memory location VEA. The result is stored in memory location VEA. If the result does not equal zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: (VEA)

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Unaffected
O Unaffected

ETXM,s,x — Extract Register From Memory (Short-Indexed)

0 7	8 11	12 15
D9	Rs	Rx

$(\overline{\text{Rs}})$ .AND. ((Rx)) → ((Rx))

The one's complement of the contents of register Rs are logically multiplied (AND function) by the contents of short-indexed memory location VEA. The result is stored in memory location VEA.

Affected: (VEA)

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Unaffected
O Unaffected

ETXB,s,x B — Extract Register From Memory (Short-Indexed) and Branch if Nonzero

0 7	8 11	12 15
DD	Rs	Rs
Virtual Branch Address		

$(\overline{\text{Rs}})$ .AND. ((Rx)) → ((Rx))
If Result does not =0,((PR)+1) → (PR)
If Result = 0,(PR)+2 → (PR)

The one's complement of the contents of register Rs is logically multiplied (AND function) by the contents of short-indexed memory location VEA. The result is stored in memory location VEA. If the result does not equal zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: (VEA)

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Unaffected
O Unaffected

OR Operand to Register (and Branch)

The following instructions logically add (OR function) bit constants and word operands contained in registers or memory to operands in the general purpose registers. The bit operation may be specified to branch unconditionally. The register word operation may be specified to branch if the result is not zero.

Appropriate Condition Codes are set if the result is zero or negative. Bit operations always set carry and clear overflow conditions. Carry and overflow are unaffected by the word operations.

OBR,r,b OR Bit in Register

0 7	8 11	12 15
63	Rd	b

$1 \rightarrow (Rd)_b$

The bit contained in bit position b in register Rd is set to one. The other bits in register Rd are not affected.

Affected: $(Rd)_b$

Condition Codes:
- Z Cleared
- N Set equal to bit 0 of result
- C Set
- O Cleared

OBRB,r,b B OR Bit in Register and Branch Unconditionally

0 7	8 11	12 15
73	Rd	B
Virtual Branch Address		

$1 \rightarrow (Rd)_b$
$((PR)+1) \rightarrow (PR)$

The bit contained in bit position b in register Rd is set to one. The other bits in register Rd are not affected. An unconditional branch is then executed to the virtual branch address.

Affected: $(Rd)_b$

Condition Codes:
- Z Cleared
- N Set equal to bit 0 of result
- C Set
- O Cleared

ORR,r,s   OR Register to Register

0 7	8 11	12 15
6B	Rd	Rs

(Rs) V (Rd) →(Rd)

The contents of register Rs are logically added to the contents of register Rd. The result is stored in register Rd.

Affected: (Rd)

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of Rd
C Unaffected
O Unaffected

ORRB,r,s B   OR Register to Register and Branch if Nonzero

0 7	8 11	12 15
7B	Rd	Rs
Virtual Branch Address		

(Rs) V (Rd) →(Rd)
If Result does not = 0,((PR)+1) →(PR)
If Result = 0,(PR)+2 →(PR)

The contents of register Rs are logically added to the contents of register Rd. The result is stored in register Rd. If the result does not equal zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: (Rd)

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of Rd
C Unaffected
O Unaffected

ORM*,r,x A   OR Memory to Register

0 7	8 11		12 15
E3	Rd	I	Rxx
Virtual Operand Address			

(VEA) V (Rd) →(Rd)

The contents of memory location VEA are logically added to the contents of register Rd. The result is stored in register Rd.

Affected: (Rd)

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of Rd
C Unaffected
O Unaffected

ORI,r V   OR Memory (Immediate) to Register

0 7	8 11	12 15
EB	Rd	0
Immediate Operand		

((PR)+1) V (Rd) →(Rd)

The contents of immediate memory location VEA are logically added to the contents of register Rd. The result is stored in register Rd.

Affected: (Rd)

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of Rd
C Unaffected
O Unaffected

ORS,r,d — OR Memory (Short-Displaced) to Register

0 7	8 11	12 15
F3	Rd	D

((R1)+D) V (Rd) → (Rd)

The contents of short-displaced memory location VEA are logically added to the contents of register Rd. The result is stored in register Rd.

Affected: (Rd)

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of Rd
C Unaffected
O Unaffected

TORI,r V — Test OR Memory (Immediate) to Register

0 7	8 11	12 15
EB	Rd	1
Immediate Operand		

Rd V ((PR)+1 → Result

The contents of immediate memory location VEA are logically added to the contents of register Rd. The result is not stored.

This instruction is not supported by the M5A assembler.

Affected: None

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of Rd
C Unaffected
O Unaffected

ORX,r,x — OR Memory (Short-Indexed) to Register

0 7	8 11	12 15
FB	Rd	Rx

((Rx)) V (Rd) → (Rd)

The contents of short-indexed memory location VEA are logically added to the contents of register Rd. The result is stored in register Rd.

Affected: (Rd)

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of Rd
C Unaffected
O Unaffected

OR Operand to Memory

The following instructions logically add (OR function) bit constants and word operands contained in the general purpose registers to operands in memory.

Appropriate Condition Codes are set if the result is zero or negative. Bit operations always set carry and clear zero and overflow conditions. Carry and overflow are unaffected by the word operations.

OBMM*,b,x A OR Bit in Memory

0 7	8 11	12	15
82	b	I	Rxx

1 → $(VEA)_b$ VOA

The bit contained in bit position b in memory location VEA is set to one. The other bits contained in the word are unaffected.

Affected: $(VEA)_b$

Condition Codes:
Z Cleared
N Set equal to bit 0 of result
C Set
O Cleared

OBSM,b,d OR Bit in Memory (Short-Displaced)

0 7	8 11	12 15
92	b	D

1 → $((R1)+D)_b$

The bit contained in bit position b in short-displaced memory location VEA is set to one. The other bits contained in the word are unaffected.

Affected: $(VEA)_b$

Condition Codes:
Z Cleared
N Set equal to bit 0 of result
C Set
O Cleared

OBXM,b,x OR Bit in Memory (Short-Indexed)

0 7	8 11	12 15
9A	b	Rx

$1 \rightarrow ((Rx))_b$

The bit contained in bit position b in short-indexed memory location VEA is set to one. The other bits contained in the word are unaffected.

Affected: $(VEA)_b$

Condition Codes:

- Z Cleared
- N Set equal to bit 0 of result
- C Set
- O Cleared

ORMM*,s,x A OR Register to Memory

0 7	8 11	12	15
C2	Rs	I	Rxx
Virtual Operand Address			

$(Rs) \vee (VEA) \rightarrow (VEA)$

The contents of register Rs are logically added to the contents of memory location VEA. The result is stored in memory location VEA.

Affected: (VEA)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Unaffected
- O Unaffected

ORSM,s,d OR Register to Memory (Short-Displaced)

0 7	8 11	12 15
D2	Rs	D

$(Rs) \vee ((R1)+D) \rightarrow ((R1)+D)$

The contents of register Rs are logically added to the contents of short-displaced memory location VEA. The result is stored in memory location VEA.

Affected: (VEA)

Condition Codes

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Unaffected
- O Unaffected

ORXM,s,x OR Register to Memory (Short-Indexed)

0 7	8 11	12 15
DA	Rs	Rx

$(Rs) \vee ((Rx)) \rightarrow ((Rx))$

The contents of register Rs are logically added to the contents of short-indexed memory location VEA. The result is stored in memory location VEA.

Affected: (VEA)

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Unaffected
- O Unaffected

Exclusive OR Operand to Register (and Branch if Nonzero)

The following instructions logically add modulo-two (Exclusive OR function) bit constants and word operands contained in the general purpose registers or memory to operands in registers. The bit and register operations may be specified to branch if the result is not zero.

Appropriate Condition Codes are set if the result is zero or negative. Bit operations set carry equal to the bit result and clear overflow. Carry and overflow are unaffected by the word operations.

XBR,r,b Exclusive OR Bit in Register

0 7	8 11	12 15
64	Rd	b

$\overline{(Rd)}_b \rightarrow (Rd)_b$

The bit contained in bit position b in register Rd is complemented. The other bits in register Rd are not affected.

Affected: $(Rd)_b$

Condition Codes
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Set to resulting state of $(Rd)_b$
- O Cleared

XBRB,r,b B Exclusive OR Bit in Register and Branch if Nonzero

0 7	8 11	12 15
74	Rd	b
Virtual Branch Address		

$\overline{(Rd)}_b \rightarrow (Rd)_b$

If Result does not = 0,((PR)+1) → (PR)

If Result = 0,(PR)+2 → (PR)

The bit contained in bit position b in register Rd is complemented. The other bits in register Rd are unaffected. If the contents of register Rd are unequal to zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: $(Rd)_b$

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Set to resulting state of $Rd)_b$
- O Cleared

XOR,r,s Exclusive OR Register to Register

0 7	8 11	12 15
6C	Rd	Rs

(Rs) .XOR. (Rd) →(Rd)

The contents of register Rs are logically added modulo-two to the contents of register Rd. The result is stored in register Rd.

Affected: (Rd)

Condition
Codes: Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Unaffected
O Unaffected

XORB,r,s B Exclusive OR Register to Register and Branch if Nonzero

0 7	8 11	12 15
7C	Rd	Rs
Virtual Branch Address		

(Rs) .XOR. (Rd) →(Rd)
If Result does not =0, ((PR)+1) →(PR)
If Result = 0,(PR)+2 →(PR)

The contents of register Rs are logically added to modulo-two to the contents of register Rd. The result is stored in register Rd. If the result does not equal zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: (Rd)

Condition
Codes: Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Unaffected
O Unaffected

XOM*,r,x A Exclusive OR Memory to Register

0 7	8 11	12	15
E4	Rd	I	Rxx
Virtual Operand Address			

(VEA) .XOR. (Rd) →(Rd)

The contents of memory location VEA are logically added modulo-two to the contents of register Rd. The result is stored in register Rd.

Affected: (Rd)

Condition
Codes: Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Unaffected
O Unaffected

XOI,r V Exclusive OR Memory to Register (Immediate)

0 7	8 11	12 15
EC	Rd	0
Immediate Operand		

((PR)+1) .XOR. (Rd) →(Rd)

The contents of immediate memory location VEA are logically added modulo-two to the contents of register Rd. The result is stored in register Rd.

Affected: (Rd)

Condition
Codes: Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Unaffected
O Unaffected

TXOI,r V — Test Exclusive OR Memory to Register (Immediate)

0 7	8 11	12 15
EC	Rd	1
Immediate Operand		

Rd .XOR. ((PR)+1 → Result

The contents of immediate memory location VEA are logically added modulo-two to the contents of register Rd. The result is not stored.

Affected: None

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Unaffected
- O Unaffected

XOS,r,d — Exclusive OR Memory to Register (Short-Displaced)

0 7	8 11	12 15
F4	Rd	D

((R1)+D) .XOR. (Rd) → (Rd)

The contents of short-indexed memory location VEA are logically added modulo-two to the contents of register Rd. The result is stored in register Rd.

Affected: (Rd)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Unaffected
- O Unaffected

XOX,r,x — Exclusive OR Memory to Register (Short-Displaced)

0 7	8 11	12 15
FC	Rd	Rx

((Rx)) .XOR. (Rd) → (Rd)

The contents of short-displaced memory location VEA are logically added modulo-two to the contents of register Rd. The result is stored in register Rd.

Affected: (Rd)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Unaffected
- O Unaffected

Zero Bit in Operand (and Branch if Nonzero)

The following instructions logically reset (Zero function) bit operands contained in the general purpose registers or memory. All operations may be specified to branch if the result is not zero.

Appropriate Condition Codes are set if the result is zero or negative. Carry and overflow cannot occur and their Condition Codes are cleared.

ZBR,r,b — Zero Bit in Register

0 7	8 11	12 15
62	Rd	b

$0 \rightarrow (Rd)_b$

The bit contained in bit position b in register Rd is cleared to zero. The other bits in register Rd are unaffected.

Affected: $(Rd)_b$

Condition Codes
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Cleared
- O Cleared

ZBRB,r,b B — Zero Bit in Register and Branch if Nonzero

0 7	8 11	12	15
81	b	I	Rxx
Virtual Operand Address			

Fel

$0 \rightarrow (Rd)_b$

If Result does not = 0, ((PR)+1) $\rightarrow$ (PR)

If Result = 0, (PR)+2 $\rightarrow$ (PR)

The bit contained in bit position b in register Rd is cleared to zero. The other bits in register Rd are unaffected. If the contents of register Rd are not equal to zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: $(Rd)_b$

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Cleared
- O Cleared

ZBMM*,b,x A — Zero Bit in Memory

0 7	8 11	12	15
81	b	I	Rxx
Virtual Operand Address			

$0 \rightarrow (VEA)_b$

The bit contained in bit position b in memory location VEA is cleared to zero. The other bits contained in the word are unaffected.

Affected: $(VEA)_b$

Condition Codes:
- Z Set if result =0, otherwise cleared
- N Set equal to bit 0 of result
- C Cleared
- O Cleared

ZBMB*,b,x A,B — Zero Bit in Memory and Branch if Nonzero

0 7	8 11	12	15
85	b	I	Rxx
Virtual Operand Address			
Virtual Branch Address			

$0 \rightarrow (VEA)_b$

If Result does not = 0,((PR)+2) $\rightarrow$ (PR)

If Result = 0,(PR)+3 $\rightarrow$ (PR)

The bit contained in bit position b in memory location VEA is cleared to zero. The other bits contained in the word are unaffected. If the resulting word is unequal to zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: $(VEA)_b$

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Cleared
- O Cleared

ZBSM,b,d — Zero Bit in Memory (Short-Displaced)

0 — 7	8 — 11	12 — 15
91	b	D

$0 \rightarrow ((R1)+D)_b$

The bit contained in bit position b in short-displaced memory location VEA is cleared to zero. The other bits contained in the word are unaffected

Affected: $(VEA)_b$

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Cleared
O Cleared

ZBSB,b,d B — Zero Bit in Memory (Short-Displaced) and Branch if Nonzero

0 — 7	8 — 11	12 — 15
95	b	D
Virtual Branch Address		

$0 \rightarrow ((R1)+D)_b$
If Result does not = 0,((PR)+1) → (PR)
If Result = 0,(PR)+2 → (PR)

The bit contained in bit position b in short-displaced memory location VEA is cleared to zero. The other bits contained in the word are unaffected. If the resulting word is unequal to zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: $(VEA)_b$

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Cleared
O Cleared

ZBXM,b,x — Zero Bit in Memory (Short-Indexed)

0 — 7	8 — 11	12 — 15
99	b	Rx

$0 \rightarrow ((Rx))_b$

The bit contained in bit position b in short-indexed memory location VEA is cleared to zero. The other bits contained in the word are unaffected.

Affected: $(VEA)_b$

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Cleared
O Cleared

ZBXB,b,x B — Zero Bit in Memory (Short-Indexed) and Branch if Nonzero

0 — 7	8 — 11	12 — 15
9D	~~D~~ b	Rx
Virtual Branch Address		

$0 \rightarrow ((Rx))_b$
If Result does not = 0,((PR)+1) → (PR)
If Result = 0, (PR)+2 → (PR)

The bit contained in bit position b in short-indexed memory location VEA is cleared to zero. The other bits contained in the word are unaffected. If the resulting word is unequal to zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: $(VEA)_b$

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Cleared
O Cleared

Transfer One's Complement Register to Register

The following instruction logically complements (One's Complement function) bit operands contained in the general purpose registers

Appropriate Condition Codes are set if the result is zero or negative. Carry and overflow cannot occur and their Condition Codes are unaffected.

TOR,r,s Transfer One's Complement Register to Register

0 — 7	8 — 11	12 — 15
0D	Rd	Rs

$(\overline{Rs}) \rightarrow (Rd)$

The one's complement of the contents of register Rs replaces the contents of register Rd.

Affected: (Rd)

Condition Codes:

Z	Set if result = 0, otherwise cleared
N	Set equal to bit 0 of result
C	Unaffected
O	Unaffected

Set Register

The following instructions are used to set and reset register Rd, depending on the combined states of N,Z,O and C. The "Less than" state is defined as only one of the negative or overflow conditions set. "Greater than or equal" is defined as neither or both of negative and overflow conditions set. "Greater than" is defined as the zero condition reset and neither or both of the negative and overflow conditions set. "Magnitude higher" is defined as the carry condition set and the zero condition reset. "Magnitude not higher" is defined as the carry condition reset or the zero condition set.

Condition codes are unaffected by these instructions.

SRNS,r — Set Register if Condition Code N Set

0 7	8 11	12 15
CA	0	Rd

if CCN=1, 0 →(Rd)
if CCN=0, -1 →(Rd)

Register Rd is set equal to 0 (True) if Condition Code N is set. Otherwise, Rd is set to -1 (False).

Affected: (Rd)

Condition Codes: Unaffected)

SRNR,r — Set Register if Condition Code N Reset

0 7	8 11	12 15
CA	8	Rd

if CCN=0, 0 →(Rd)
if CCN=1, -1 →(Rd)

Register Rd is set equal to 0 (True) if Condition Code N is reset. Otherwise, Rd, is set to -1 (False).

Affected: (Rd)

Condition Codes: Unaffected

SRZS,r — Set Register if Condition Code Z Set

0 7	8 11	12 15
CA	1	Rd

if CCZ=1, 0 →(Rd)
if CCZ=0, -1 →(Rd)

Register Rd is set equal to 0 (True) if Condition Code Z is set. Otherwise, Rd is set to -1 (False).

Affected: (Rd)

Condition Codes: Unaffected

SRZR,r — Set Register if Condition Code Z Reset

0 7	8 11	12 15
CA	9	Rd

if CCZ=0, 0 →(Rd)
if CCZ=1, -1 →(Rd)

Register Rd is set equal to 0 (True) if Condition Code Z is reset. Otherwise, Rd is set to -1 (False).

Affected: (Rd)

Condition Codes: Unaffected

SROS,r — Set Register if Condition Code O Set

0 7	8 11	12 15
CA	2	Rd

if CCO=1, 0 →(Rd)
if CCO=0, -1 →(Rd)

Register Rd is set equal to 0 (True) if Condition Code O is set. Otherwise, Rd is set to -1 (False).

Affected: (Rd)

Condition Codes: Unaffected

SROR,r — Set Register if Condition Code O is Reset

0 7	8 11	12 15
CA	A	Rd

if CCO=0, 0 →(Rd)
if CCO=1, -1 →(Rd)

Register Rd is set equal to 0 (True) if Condition Code O is reset. Otherwise, Rd is set to -1 (False)

Affected: (Rd)

Condition Codes: Unaffected

SRCS,r — Set Register if Condition Code C Set

0 7	8 11	12 15
CA	3	Rd

if CCC=1, 0 →(Rd)
if CCC=0, -1 →(Rd)

Register Rd is set equal to 0 (True) if Condition Code C is set. Otherwise, Rd is set to -1 (False).

Affected: (Rd)

Condition Codes: Unaffected

SRCR,r — Set Register if Condition Code C Reset

0 7	8 11	12 15
CA	B	Rd

if CCC=0, 0 →(Rd)
if CCC=1, -1 →(Rd)

Register Rd is set equal to 0 (True) if Condition Code is reset. Otherwise, Rd is set to -1 (False).

Affected: (Rd)

Condition Codes: Unaffected

SRGE,r — Set Register on Greater Than Or Equal Condition

0 7	8 11	12 15
CA	C	Rd

if CCO .XOR. CCN=0, 0 →(Rd)
if CCO .XOR. CCN=1, -1 →(Rd)

If neither or both of negative and overflow conditions are set, the preceding instruction's destination operand is algebraically greater than or equal to the source operand, and Rd is set to equal to 0 (True). Otherwise Rd is set equal to -1 (False).

Affected: (Rd)

Condition Codes: Unaffected

SRGT,r — Set Register on Greater Than Condition

0 7	8 11	12 15
CA	D	Rd

if CCZ V (CCN .XOR. CCO) = 0, 0 →(Rd)
if CCZ V (CCN .XOR. CCO) = 1, -1 →(Rd)

If the zero condition is reset and neither or both of the negative and overflow conditions are set, the preceding instruction's destination operand is algebraically greater than the source operand, and register Rd is set equal to 0 (True). Otherwise, Rd is set to -1 (False).

Affected: (Rd)

Condition Codes: Unaffected

SRLE,r — Set Register on Less Than or Or Equal Condition

0 7	8 11	12 15
CA	5	Rd

if CCZ V (CCN .XOR. CCO) =1, 0 →(Rd)
if CCZ V (CCN .XOR. CCO) = 0, -1 →(Rd)

If the zero condition is set or only one of the negative or overflow conditions is set, the previous instruction's destination operand is algebraically less than or equal to the source operand, and register Rd is set equal to 0 (True). Otherwise, Rd is set to -1 (False).

Affected: (Rd)

Condition Codes: Unaffected

SRLS,r — Set Register on Less Than Condition

0 7	8 11	12 15
CA	4	Rd

if CCO .XOR. CCN=1, 0 →(Rd)
if CCO .XOR. CCN=0, -1 →(Rd)

If only one of the Negative or Overflow conditions is set, the preceding instruction's destination operand is algebraically less than the source operand and the register Rd is set equal to 0 (True). Otherwise, Rd is set to -1 (False).

Affected: (Rd)

Condition Codes: Unaffected

SRHI,r Set Register on Magnitude Higher Condition

0 7	8 11	12 15
CA	6	Rd

if CCC .AND. $\overline{CCZ}$=1, 0 → (Rd)
if CCC .AND. $\overline{CCZ}$=0, -1 → (Rd)

If the carry condition is set and the zero condition is not, the previous instruction's destination operand was higher in magnitude than the source operand, and Rd is set equal to 0 (True). Otherwise, Rd is sèt to -1 (False).

Affected: (Rd)

Condition Codes: Not affected

SRNH,r Set Register on Magnitude Not Higher Condition

0 7	8 11	12 15
CA	E	Rd

if $\overline{CCC}$ V CCZ=1, 0 → (Rd)
if $\overline{CCC}$ V CCZ=0, -1 → (Rd)

If the carry condition is reset or the zero condition is set, the previous instruction's destination operand was not higher in magnitude than the source operand, and Rd is set equal to 0 (True). Otherwise Rd is set to -1 (False).

Affected: (Rd)

Condition Codes: Not affected

Shift Instructions

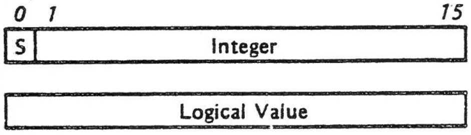
This instruction class provides the capability to reposition bits left or right within single, double or quadruple-registers. All combinations of arithmetic or logical, left or right, and single, double or quadruple-register shift operations are provided. In all operations, the number of bit positions that may be shifted ranges from zero (shift count equal 0) to fifteen (shift count equal 15).

In addition, a one-bit-left-rotate single-register operation is included. All of the instructions in this class set Condition Codes.

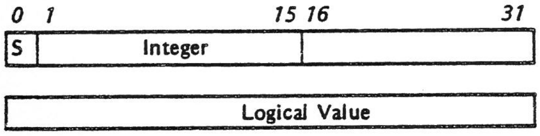
In all multiple-register shift operations, the register specified by the instruction word must be the even-number register that contains the most significant part of the logical or arithmetic operand. Improper register designation or operand positioning will result in either high-order or low-order positions of the operand not being shifted.

The following signed integer (arithmetic) and logical data formats are valid for respective word, double-word and quadruple-word operands in shift operations.

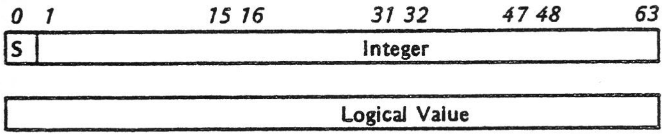
Single-register R, where: R=0–15



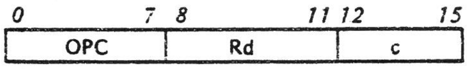
Double-register R:RV1, where: R=2, 4, 6, 8, 10, 12 or 14



Quadruple-register R:RV1:RV2:RV3, where:R=4, 8 or 12



The execution of each shift instruction will shift the operand zero to 15 bit positions, as defined by the binary coded shift count field (bits 12-15) in the instruction word. (The rotate instruction circular shifts just one bit position and a shift count is not specified.) The special format of the Shift Instruction is illustrated below.

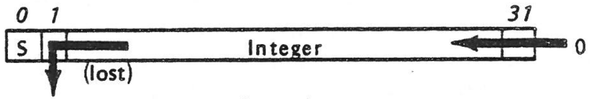


Where:

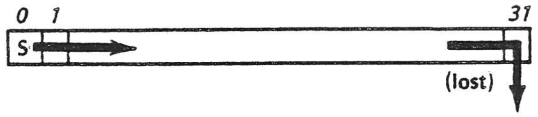
OPC = shift operation code
Rd = shift register(s)
c = shift count (number of bits to shift - i.e., 0-15)

In an arithmetic left shift and in all cases of a logical left or right shift, "c" zero (0) bits are shifted into the vacated bit positions and bits shifted out of the registers are lost. In an arithmetic right shift, the sign bit is propagated into all vacated high order bit positions. These properties are illustrated as follows. Note that in arithmetic shifts the high-order sign bit (bit 0) is unaffected.

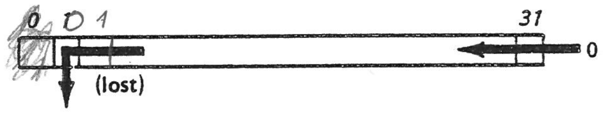
Arithmetic Left Shift (c bits)



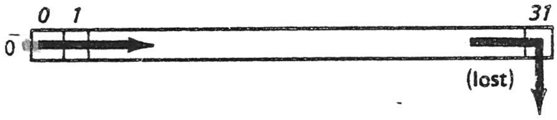
Arithmetic Right Shift (c bits)



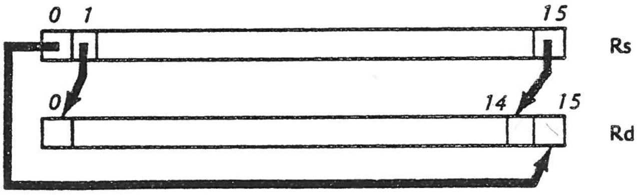
Logical Left Shift (c bits)



Logical Right Shift (c bits)



Left Rotate (1 bit)



Shift Arithmetic Operand in Register(s)

The following instructions arithmetically shift word, double-word, or quadruple-word operands contained in the general purpose registers. The number of bit positions specified (0-15) are shifted left or right. Left shifts result in high-order bit(s) truncation (excluding the sign bit) and low-order bit(s) zero-fill, and perform the function of positive scaling a fixed point value or of increasing an integer value by a power-of-two. Right shifts result in high-order sign extension and low-order bit(s) truncation, and perform the function of negative scaling a fixed point value or of decreasing an integer value by a power-of-minus-two.

Appropriate Condition Codes are set if the result is zero or negative. The carry condition is set to the value of the last bit shifted out of the register(s). Overflow is set during a left shift if any bit shifted out of the most significant magnitude bit position (bit 1) differs from the sign bit (bit 0)

LAS,r,c — Shift Left Arithmetic Single-Register

0 7	*8 11*	*12 15*
2F	Rd	c

$(Rd)_{(1+c)-15} \rightarrow (Rd)_{1-(15-c)}$

$0\text{'s} \rightarrow (Rd)_{(1+15-c)-15}$

The contents of register Rd are shifted left arithmetically c bit positions.

Affected: (Rd)

Condition Codes

- Z Set if result = 0, otherwise cleared.
- N Set equal to bit 0 of result
- C Contains bit last shifted from the register; zero if count equals zero
- O Set if most significant bit (1) differs from the sign bit (0) before the shift count = 0

RAS,r,c — Shift Right Arithmetic Single-Register

0 7	*8 11*	*12 15*
2B	Rd	c

$(Rd)_{1-(15-c)} \rightarrow (Rd)_{(1+c)-15}$

$(Rd)_0 \rightarrow (Rd)_{1-c}$

The contents of register Rd are shifted right arithmetically c bit positions.

Affected: *(Rd)*

Condition Codes:

- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Contains bit last shifted from the register; zero if count equals zero
- O Cleared

LAD,r,c — Shift Left Arithmetic Double-Register

0 7	8 11	12	15
2E	Rd	0	c

$$(Rd{:}RdV1)_{(1+c)-31} \longrightarrow (Rd{:}RdV1)_{1-(31-c)}$$
$$0\text{'s} \longrightarrow (RdV1)_{(1+15-c)-15}$$

The contents of registers Rd:RdV1 are shifted left arithmetically c bit positions.

Affected: (Rd:RdV1)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Contains bit last shifted from the register; zero if count equals zero
- O Set if most significant bit (1) differs from the sign bit (0) before the shift count = 0

RAD,r,c — Shift Right Arithmetic Double-Register

0 7	8 11	12	15
2A	Rd	0	c

$$(Rd{:}RdV1)_{(1+c)-31} \longrightarrow (Rd{:}RdV1)_{1-(31-c)}$$
$$0\text{'s} \longrightarrow (RdV3)_{1-c}$$

The contents of registers Rd:RdV1 are shifted right arithmetically c bit positions.

Affected: (Rd:RdV1)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Contains bit last shifted from the register; zero if count equals zero
- O Set if most significant bit (1) differs from the sign bit (0) before the shift count = 0

LAQ,r,c — Shift Left Arithmetic Quadruple-Register

0 7	8	11	12	15
2E	Rd	0	1	c

$$(Rd{:}RdV1{:}RdV2{:}RdV3)_{(1+c)-63} \longrightarrow (Rd{:}RdV1{:}RdV2{:}RdV3)_{1-(63-c)}$$
$$0\text{'s} \longrightarrow (RdV3)_{(1+15-c)-15}$$

The contents of registers Rd:RdV1:RdV2:RdV3 are shifted left arithmetically c bit positions.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Contains bit last shifted from the register; zero if count equals zero
- O Set if most significant bit (1) differs from the sign bit (0) before the shift count = 0

RAQ,r,c — Shift Right Arithmetic Quadruple-Register

0 7	8	11	12	15
2A	Rd	0	1	c

$$(Rd{:}RdV1{:}RdV2{:}RdV3)_{1-(63-c)} \longrightarrow (Rd{:}RdV1{:}RdV2{:}RdV3)_{(1-c)-63}$$
$$(Rd)_0 \longrightarrow (Rd)_{1-c}$$

The contents of registers Rd:RdV1:RdV2:RdV3 are shifted right arithmetically c bit position.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Contains bit last shifted from the register; zero if count equals zero
- O Cleared

Shift Logical Operand in Register(s)

The following instructions logically shift word, double-word or quadruple-word operands contained in the general purpose registers. The number of bit positions specified (0-15) are shifted left or right. Left shifts result in high-order bit(s) truncation (including the sign bit) and low-order bit(s) zero-fill. Right shifts result in high-order bit(s) zero-fill and low-order bit(s) truncation.

Appropriate Condition Codes are set if the result is zero or negative. The carry condition is set to the value of the last bit shifted out of the register(s), and overflow is always cleared.

LLS,r,c — Shift Left Logical Single-Register

0 7	8 11	12 15
2D	Rd	c

$(Rd)_{c-15} \rightarrow (Rd)_{0-(15-c)}$

$0\text{'s} \rightarrow (Rd)_{(1+15-c)-15}$

The contents of register Rd are shifted left logically c bit positions.

Affected: (Rd)

Condition Codes:

Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Contains last bit shifted from the register; zero if shift count equals zero
O Cleared

RLS,r,c — Shift Right Logical Single-Register

0 7	8 11	12 15
29	Rd	c

$(Rd)_{0-(15-c)} \rightarrow (Rd)_{c-15}$

$0\text{'s} \rightarrow (Rd)_{0-(c-1)}$

The contents of register Rd are shifted right logically c bit positions.

Affected: (Rd)

Condition Codes:

Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Contains last bit shifted from the register; zero if shift count equals zero
O Cleared

LLD,r,c — Shift Left Logical Double-Register

0 7	8	11	12 15
2C	Rd	0	c

$(\mathrm{Rd{:}RdV1})_{c-31} \rightarrow (\mathrm{Rd{:}RdV1})_{0-(31-c)}$

$0\text{'s} \rightarrow (\mathrm{RdV1})_{(1+15-c)-15}$

The contents of registers Rd:RdV1 are shifted left logically c bit positions.

Affected: (Rd:RdV1)

Condition Codes:

Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Contains last bit shifted from the register, zero if shift count equals zero
O Cleared

RLD,r,c — Shift Right Logical Double-Register

0 7	8	11	12 15
28	Rd	0	c

$(\mathrm{Rd{:}RdV1})_{0-(31-c)} \rightarrow (\mathrm{Rd{:}RdV1})_{c-31}$

$0\text{'s} \rightarrow (\mathrm{Rd})_{0-(c-1)}$

The contents of registers Rd:RdV1 are shifted right logically c bit positions.

Affected: (Rd:RdV1)

Condition Codes:

Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Contains last bit shifted from the register, zero if shift count equals zero
O Cleared

LLQ,r,c — Shift Left Logical Quadruple-Register

0 7	8		11	12 15
2C	Rd	0	1	c

$(\mathrm{Rd{:}RdV1{:}RdV2{:}RdV3})_{c-63} \rightarrow (\mathrm{Rd{:}RdV1{:}RdV2{:}RdV3})_{0-(63-c)}$

$0\text{'s} \rightarrow (\mathrm{RdV3})_{(1+15-c)-15}$

The contents of registers Rd:RdV1:RdV2:RdV3 are shifted left logically c bit positions.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:

Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Contains last bit shifted from the register, zero if shift count equals zero
O Cleared

RLQ,r,c — Shift Right Logical Quadruple-Register

0 7	8		11	12 15
28	Rd	0	1	c

$(\mathrm{Rd{:}RdV1{:}RdV2{:}RdV3})_{0-(63-c)} \rightarrow (\mathrm{Rd{:}RdV1{:}RdV2{:}RdV3})_{c-63}$

$0\text{'s} \rightarrow (\mathrm{Rd})_{0-(c-1)}$

The contents of registers Rd:RdV1:RdV2:RdV3 are shifted right logically c bit positions.

Affected: (Rd:RdV1:RdV2:RdV3)

Condition Codes:

Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Contains last bit shifted from the register, zero if shift count equals zero
O Cleared

Rotate Word in Register

The following instruction circular shifts (rotates) a word operand contained in the general purpose registers one bit position to the left and places the bit shifted out of bit position 0 back into bit position 15.

Appropraite Condition Codes are set if the result is zero or negative. The carry condition is set to the value of the bit shifted out of bit position 0, and overflow is always cleared.

LRS,r,s — Left Rotate Single-Register to Register

0 – 7	8 – 11	12 – 15
0F	Rd	Rs

$(Rs)_{1-15} \rightarrow (Rd)_{0-14}$
$(Rs)_{0} \rightarrow (Rd)_{15}$

The contents of register Rd are replaced by the contents of register Rs shifted left one bit position. Bit position 0 of register Rs is rotated into bit position 15 of register Rd.

Affected: (Rd)

Condition Codes:
Z Set if result = 0, otherwise cleared
N Set equal to bit 0 of result
C Contains bit 15 of result
O Cleared

Compare and Test Instructions

This instruction class provides the capability to perform arithmetic and unsigned magnitude comparisons of bit, word, double, triple, and quad-word operands, plus logical testing (AND function) of bit. The operands involved in these comparison or test operations are not altered.

Comparisons may be made between word, double, triple, or quad-word operands in the general purpose registers, and memory or between bit constants and word operands in memory. The memory reference instructions include all standard addressing modes. All indexed instructions may address memory operands via extended addressing. All of the compare bit to memory and most of the compare register to memory operations provide branch addresses to be used if the comparison produces an equal or a less-than result.

Test instructions test the settings of individual bits or the bit matching of words. Operands for test instructions may be in registers or in memory. The memory referencing instructions include all standard addressing modes.

All of the test operations provide a branch address to be used if the result contains any one's. All compare and test instructions set Condition Codes.

To supplement the explicit branch addresses in many of these instructions, Condition Codes set by any compare or test instruction may be tested by subsequent Branch or Hop instructions to effect program branching on a variety of signed arithmetic, unsigned magnitude or bit logical relationships between compared (or tested) operands.

Compare Register(s) to Operand (and Branch Equal or Less)

The following instructions compare operands contained in the general purpose registers to operands contained in registers or memory. Comparison is performed by algebraic subtraction, i.e., the contents of (extended) register Rd minus (compared to) the source operand. The contents of the registers and memory remain unchanged. Most memory operations may be specified to branch on an equal or less than result.

Overflow will occur when operands of opposite sign produce a result with the same sign as register Rs or memory location VEA.

Appropriate Condition Codes are set if the comparative result is zero or negative, or to indicate adder carry out or overflow.

CRR,r,s Compare Register to Register

0 7	8 11	12 15
6E	Rd	Rs

(Rd)-(Rs) ⟶Result

The contents of register Rs are algebraically subtracted from the contents of register Rd. The result is not stored.

Affected: None

Condition Codes:
- Z Set if operands are equal (result = 0), otherwise cleared
- N Set if bit 0 of Result = 1, otherwise cleared
- C Set if carry from most significant bit of result, otherwise cleared
- O Set if operands were of opposite signs and the sign of the second operand is the same as the result, otherwise cleared

CRRD,r,s Compare Double-Register to Double-Register

0 7	8 11		12 15	
CF	Rd	0	Rs	0

(Rd:RdV1)-(Rs:RsV1) ⟶Result

The contents of the registers Rs:RsV1 are algebraically subtracted from the contents of registers Rd:RdV1. The result is not stored.

Affected: None

Condition Codes:
- Z Set if operands are equal (result = 0), otherwise cleared
- N Set if bit 0 of Result = 1, otherwise cleared
- C Set if carry from most significant bit of result, otherwise cleared
- O Set if operands were of opposite signs and the sign of the second operand is the same as the result, otherwise cleared

CRRT,r,s Compare Triple-Register to Triple-Register

0 7 8 9 10 11 12 15

8B	Rd	1	0	Rs	0	0

(Rd:RdV1:RdV2) - (Rs:RsV1:RsV2) ⟶Result

The contents of register Rs:RsV1:RsV2 are subtracted from the contents of register Rd:RdV1:RdV2. The condition codes are loaded, but the result is not.

Affected: None

Condition Codes:
- Z Set if operands are equal (result = 0), other-
- N wise cleared
- N Set if bit 0 of Result = 1, otherwise cleared
- C Set if carry from most significant bit of result, otherwise cleared
- O Set if operands were of opposite signs and the sign of the second operand is the same as the result, otherwise cleared

CRRQ,r,s Compare Quad-Register to Quad-Register

0 7 8 11 12 15

8B	Rd	1	1	Rs	0	0

(Rd:RdV1:RdV2:RdV3)-(Rs:RsV1:RsV2:RsV3) ⟶Result

The contents of registers Rs:RsV1:RsV2:RsV3 are algebraically subtracted from the contents of Rd:RdV1:RdV2:RdV3. The condition codes are loaded, but the result is not.

Affected: None

Condition Codes:
- Z Set if operands are equal (result = 0), otherwise cleared
- N Set if bit 0 of Result = 1, otherwise cleared
- C Set if carry from most significant bit of result, otherwise cleared
- O Set if operands were of opposite signs and the sign of the second operand is the same as the result, otherwise cleared

CRM*,r,x A Compare Register to Memory

0 7	8 11	12	15
C3	Rd	I	Rxx
Virtual Operand Address			

(Rd) - (VEA) → Result

The contents of memory location VEA are algebraically subtracted from the contents of register Rd. The result is not stored.

Affected: None

Condition Codes:

- Z Set if operands are equal (result = 0), otherwise cleared
- N Set if bit 0 of Result = 1, otherwise cleared
- C Set if carry from most significant bit of result, otherwise cleared
- O Set if operands were of opposite signs and the sign of the second operand is the same as the result, otherwise cleared

CRMB,r,x A,B,B Compare Register to Memory and Branch if Equal or Less

0 7	8 11	12	15
C7	Rd	I	Rxx
Virtual Operand Address			
Virtual Branch Address=			
Virtual Branch Address<			

(Rd) - (VEA) → Result
If (Rd)=(VEA),((PR)+2) → (PR)
If (Rd)<(VEA),((PR)+3) → (PR)
If (Rd)<(VEA),(PR)+4 → (PR)

The contents of memory location VEA are algebraically subtracted from the contents of register Rd. The result is not stored. If the result equals zero, the operands are equal and a branch is executed to the first virtual branch address. If the result is negative, the register is less and a branch is executed to the second virtual branch address. Otherwise, the register is greater and the next instruction in sequence is executed.

Affected: None

Condition Codes:

- Z Set if operands are equal (result = 0), otherwise cleared
- N Set if bit 0 of Result = 1, otherwise cleared
- C Set if carry from most significant bit of result, otherwise cleared
- O Set if operands were of opposite signs and the sign of the second operand is the same as the result, otherwise cleared

CRI,r V Compare Register to Memory (Immediate)

0 7	8 11	12 15
E9	Rd	1
Immediate Operand		

(Rd) - ((PR)+1) → Result

The contents of immediate memory location VEA are algebraically subtracted from the contents of register Rd. The result is not stored.

Affected: None

Condition Codes:

- Z Set if operands are equal (result = 0), otherwise cleared
- N Set if bit 0 of Result = 1, otherwise cleared
- C Set if carry from most significant bit of result, otherwise cleared
- O Set if operands were of opposite signs and the sign of the second operand is the same as the result, otherwise cleared

CRS,r,d Compare Register to Memory (Short-Displaced)

0 7	8 11	12 15
D3	Rd	D

(Rd - ((R1)+D) → Result

The contents of short-displaced memory location VEA are algebraically subtracted from the contents of register Rd. The result is not stored.

Affected: None

Condition Codes:

- Z Set if operands are equal (result = 0), otherwise cleared
- N Set if bit 0 of Result = 1, otherwise cleared
- C Set if carry from most significant bit of result, otherwise cleared
- O Set if operands were of opposite signs and the sign of the second operand is the same as the result, otherwise cleared

CRSB,r,d B,B — Compare Register to Memory (Short-Displaced) and Branch If Equal or Less

0 7	8 11	12 15
D7	Rd	D
Virtual Branch Address=		
Virtual Branch Address <		

```
             (Rd) - ((R1+D)  → Result
If (Rd) = ((R1)+D),((PR)+1)  → (PR)
If (Rd) < ((R1)+D),((PR)+2)  → (PR)
 If (Rd) > ((R1)+D),(PR)+3   → (PR)
```

The contents of short-displaced memory location VEA are algebraically subtracted from the contents of register Rd. The result is not stored. If the result equals zero, the operands are equal and a branch is executed to the first virtual branch address. If the result is negative the register is less and a branch is executed to the second virtual branch address. Otherwise the register is greater and the next instruction in sequence is executed.

Affected: None

Condition Codes:

Z Set if operands are equal (result = 0), otherwise cleared

N Set if bit 0 of Result = 1, otherwise cleared

C Set if carry from most significant bit of result, otherwise cleared

O Set if operands were of opposite signs and the sign of the second operand is the same as the result, otherwise cleared

CRX,r,x — Compare Register to Memory (Short-Indexed

0 7	8 11	12 15
DB	Rd	Rx

(Rd) - ((Rx)) → Result

The contents of short-indexed memory location VEA are algebraically subtracted from the contents of register Rd. The result is not stored.

Affected: None

Condition Codes:

Z Set if operands are equal (result = 0), otherwise cleared

N Set if bit 0 of Result = 1, otherwise cleared

C Set if carry from most significant bit of result, otherwise cleared

O Set if operands were of opposite signs and the sign of the second operand is the same as the result, otherwise cleared

CRXD,r,x Compare Double Register to (Short-Indexed) Memory Doubleword

0 7	8 10	11	12 15
8E	Rd	0	Rx

(Rd:RdV1)-(VEA:VEA+1) ⟶Result

The contents of memory locations VEA:VEA+1 are algebraically subtracted from the contents of register Rd:RdV1. The condition codes are loaded, but the result is not.

Affected: None

Condition Codes:

- Z Set if operands are equal (result = 0), otherwise cleared
- N Set if bit 0 of Result = 1, otherwise cleared
- C Set if carry from most significant bit of result, otherwise cleared
- O Set if operands were of opposite signs and the sign of the second operand is the same as the result, otherwise cleared

CRXB,r,x B,B Compare Register to Memory (Short-Indexed) and Branch If Equal or Less

0 7	8 11	12 15
DF	Rd	Rx
Virtual Branch Address=		
Virtual Branch Address<		

```
                (Rd) - ((Rx))      ⟶Result
If (Rd) =((RX)),((PR)+1)          ⟶(PR)
If (Rd)<((Rx)),((PR)+2)           ⟶(PR)
If (Rd) > ((Rx)),(PR)+3           ⟶(PR)
```

The contents of short-indexed memory location VEA are algebraically subtracted from the contents of register Rd. The result is not stored. If the result equals zero, the operands are equal and a branch is executed to the first virtual branch address. If the result is negative, the register is less and a branch is exeucted to the second virtual branch address. Otherwise, the register is greater and the next instruction in sequence is executed.

Affected: None

Condition Codes:

- Z Set if operands are equal (result = 0), otherwise cleared
- N Set if bit 0 of Result = 1, otherwise cleared
- C Set if carry from most significant bit of result, otherwise cleared
- O Set if operands were of opposite signs and the sign of the second operand is the same as the result, otherwise cleared

CRMD*,r,x A Compare Double-Register to Memory Doubleword

0 7	8 11	12 15
CF	Rd	I Rxx
Virtual Operand Address		

(Rd:RdV1) – (VEA:VEA+1) ⟶Result

The contents of memory locations VEA:VEA+1 are algebraically subtracted from the contents of registers Rd:RdV1. The result is not stored.

Affected: None

Condition Codes:

- Z Set if operands are equal (result = 0), otherwise cleared
- N Set if bit 0 of Result = 1, otherwise cleared
- C Set if carry from most significant bit of result, otherwise cleared
- O Set if operands were of opposite signs and the sign of the second operand is the same as the result, otherwise cleared

CRZ,s Compare Register to Zero

0 7	8 11	12 15
E8	Rs	9

((Rs) ⟶Result

The contents of register Rs are compared to zero. The condition codes are loaded.

Affected: None

Condition Codes:

- Z Set if operands are equal (result = 0), otherwise cleared
- N Set if bit 0 of Result = 1, otherwise cleared
- C Set if carry from most significant bit of result, otherwise cleared
- O Set if operands were of opposite signs and the sign of the second operand is the same as the result, otherwise cleared

CRZD,s Compare Double Register to Zero

0 7	8 11	12 15	
E8	Rs	0	A

(Rs:RsV1) ⟶Result

The contents of register Rs:RsV1 are compared to zero. The condition codes are loaded.

Affected: None

Condition Codes:

- Z Set if operands are equal (result = 0), otherwise cleared
- N Set if bit 0 of Result = 1, otherwise cleared
- C Set if carry from most significant bit of result, otherwise cleared
- O Set if operands were of opposite signs and the sign of the second operand is the same as the result, otherwise cleared

Compare Bit to Memory (and Branch)

The following instructions compare bit constants (power-of-two) to word operands in memory. Comparison is performed by algebraic subtraction, i.e., a power-of-two minus (compared to) the source memory location VEA. The contents of memory remain unchanged. All operations will branch on an equal or less than result.

Overflow will occur when operands of opposite sign produce a result with the same sign as memory location VEA.

Appropriate Condition Codes are set if the comparative result is zero or negative or to indicate adder carry out or overflow.

CBMB*,b,x   A,B,B   Compare Bit to Memory and Branch if Equal or Less

<table>
<tr><td><i>0 7</i></td><td><i>8 11</i></td><td colspan="2"><i>12 15</i></td></tr>
<tr><td>87</td><td>b</td><td>I</td><td>Rxx</td></tr>
<tr><td colspan="4">Virtual Operand Address</td></tr>
<tr><td colspan="4">Virtual Branch Address=</td></tr>
<tr><td colspan="4">Virtual Branch Address<</td></tr>
</table>

2^{15-b}-(VEA) → Result
If 2^{15-b}=(VEA),((PR)+2) → (PR)
If 2^{15-b}<(VEA),((PR)+3) → (PR)
If 2^{15-b}>(VEA),(PR)+4 → (PR)

The contents of memory location VEA are algebraically subtracted from the value +2^{15-b}. The result is not stored. If the result equals zero, the operands are equal and a branch is executed to the first virtual branch address. If the result is negative, the power-of-two is less and a branch is executed to the second virtual branch address. Otherwise, the power of two is greater and the next instruction in sequence is executed.

Affected: None

Condition Codes:

- Z Set if operands are equal (Result =0) otherwise cleared
- N Set if first operand $2^{(15-b)}$ is less than (Bit 0 of result = 1), otherwise cleared
- C Set if carry from most significant bit of result, otherwise cleared
- O Set if operands were of opposite signs and the sign of the second operand (VEA) was the same as the result, otherwise cleared

CBSB,b,d   B,B   Compare Bit to Memory (Short-Displaced) and Branch If Equal or Less

<table>
<tr><td><i>0 7</i></td><td><i>8 11</i></td><td><i>12 15</i></td></tr>
<tr><td>97</td><td>b</td><td>D</td></tr>
<tr><td colspan="3">Virtual Branch Address=</td></tr>
<tr><td colspan="3">Virtual Branch Address<</td></tr>
</table>

2^{15-b}-((R1)+D) → Result
If 2^{15-b}=((R1)+D),((PR)+1) → (PR)
If 2^{15-b}<((R1)+D),((PR)+2) → (PR)
If 2^{15-b}>((R1)+D),(PR)+3 → (PR)

The contents of short-displaced memory location VEA are algebraically subtracted from the value 2^{15-b}. If the result equals zero, the operands are equal and a branch is executed to the first virtual branch address. If the result is negative, the power-of-two is less and a branch is executed to the first virtual branch address. Otherwise, the power-of-two is greater and the next instruction in sequence is executed.

Affected: None

Condition Codes:

- Z Set if operands are equal (Result = 0) otherwise cleared
- N Set if first operand $2^{(15-b)}$ is less than (Bit 0 of result = 1), otherwise cleared
- C Set if carry from most significant bit of result, otherwise cleared
- O Set if operands were of opposite signs and the sign of the second operand (VEA) was the same as the result, otherwise cleared

CBXB,b,x B,B Compare Bit to Memory (Short-Indexed) and Branch If Equal Or Less

0 7	8 11	12 15
9F	b	Rx
Virtual Branch Address=		
Virtual Branch Address<		

2^{15-b}-((Rx)) → Result
If 2^{15-b}=((Rx)),((PR)+1 → (PR)
If 2^{15-b} <((Rx)),((PR)+2) → (PR)
If 2^{15-b} >((Rx)),(PR)+3 → (PR)

The contents of short-indexed memory location VEA are algebraically subtracted from the value 2^{15-b}. If the result equals zero, the operands are equal and a branch is executed to the first virtual branch address. If the result is negative, the power-of-two is less and a branch is executed to the second virtual branch address. Otherwise, the power-of-two is greater and the next instruction in sequence is executed.

Affected: None

Condition Codes:

- Z Set if operands are equal (Result =0) otherwise cleared
- N Set if first operand $2^{(15-b)}$ is less than (Bit 0 of result = 1), otherwise cleared
- C Set if carry from most significant bit of result, otherwise cleared
- O Set if operands were of opposite signs and the sign of the second operand (VEA) was the same as the result, otherwise cleared

Test Register and Operand (and Branch if any Ones)

The following instructions test word operands contained in the general purpose registers with bit constants or operands contained in registers or memory. Bit testing is performed on the contents of register Rd. Word testing is performed by logical product, i.e., register Rd AND the source operand. The contents of the registers and memory remain unchanged. All operations but one will branch if the logical result contains any ones.

Appropriate Condition Codes are set if register Rd (for bit operations) or the logical result (for word operations) contains all zero bits or if bit 0 is one. Bit operations will indicate carry if the bit tested is one.

TBR,r,b Test Bit(s) in Register

0 7	8 11	12 15
66	Rd	b

(Rd) → Result

The contents of register Rd are tested.

Affected: None

Condition Codes:
- Z Set if (Rd) = 0, otherwise cleared
- N Set equal to bit 0 of Rd
- C Set equal to bit b of Rd
- O Cleared

TBRB,r,b B Test Bit(s) in Register and Branch if One

0 7	8 11	12 15
76	Rd	b
Virtual Branch Address		

(Rd) → Result
If $(Rd)_b$= 1,((PR)+1) → (PR)
If $(Rd)_b$=0,(PR)+2 → (PR)

The contents of register Rd are tested. If the value in bit position b of register Rd is equal to one, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes:
- Z Set if (Rd) = 0, otherwise cleared
- N Set equal to bit 0 of Rd
- C Set equal to bit b of Rd
- O Cleared

TERB,r,s B — Test Register and Register and Branch if any Ones Compare

0 — 7	8 — 11	12 — 15
7E	Rd	Rs
Virtual Branch Address		

```
                         (Rs) .AND. (Rd)         →Result
If (Rs) .AND. (Rd) does not = 0,((PR)+1)        →(PR)
If (Rs) .AND. (Rd) = 0,(PR)+2                   →(PR)
```

The contents of register Rs are logically multiplied (AND function) by the contents of register Rd. The result is not stroed. If the result does not equal zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Unaffected
- O Unaffected

TRMB*,r,x A,B — Test Register and Memory and Branch if any Ones Compare

0 — 7	8 — 11	12	13 — 15
C6	Rd	I	Rxx
Virtual Operand Address			
Virtual Branch Address			

```
                         (Rd) .AND. (VEA)        →Result
If (Rd) .AND. (VEA) does not = 0,((PR)+2)       →(PR)
         If (Rd) .AND. (VEA) = 0, (PR)+3        →(PR)
```

The contents of short-displaced memory location VEA are logically multipled (AND function) by the contents of register Rd. The result is not stored. If the result does not equal zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Unaffected
- O Unaffected

TRSB,r,d B — Test Register and Memory (Short-Displaced) and Branch if any Ones Compare

0 — 7	8 — 11	12 — 15
D6	Rd	D
Virtual Branch Address		

```
                         (Rd) .AND. ((R1)+D)     →Result
If (Rd) .AND. ((R1)+D) does not = 0, ((PR)+1)   →(PR)
If (Rd) .AND. ((R1)+D) = 0, (PR)+2              →(PR)
```

The contents of memory location VEA are logically multiplied (AND function) by the contents of register Rd. The result is not stored. If the result does not equal zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Unaffected
- O Unaffected

TRXB,r,x B — Test Register and Memory (Short-Indexed) and Branch if any Ones Compare

0 — 7	8 — 11	12 — 15
DE	Rd	Rx
Virtual Branch Address		

```
                         (Rd) .AND. ((Rx))       →Result
If (Rd) .AND. ((Rx)) does not = 0, ((PR)+1)     →(PR)
If (Rd) .AND. ((Rx)) = 0, (PR)+2                →(PR)
```

The contents of short-indexed memory location VEA are logically multiplied (AND function) by the contents of register Rd. The result is not stored. If the result does not equal zero, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes:
- Z Set if result = 0, otherwise cleared
- N Set equal to bit 0 of result
- C Unaffected
- O Unaffected

Test Bit(s) in Memory (and Branch if One)

The following instructions test word operands contained in memory with bit constants. Testing is performed on the unaltered contents of memory. Some operations may be specified to branch if the designated bit is one. TSBM sets bit "b" after testing the addressed operand.

TBMM*,b,x A Test Bit(s) in Memory

0 7	8 11	12	13 15
83	b	I	Rxx
Virtual Operand Address			

(VEA) → Result

The contents of memory location VEA are tested.

Affected: None

Condition Codes
Z Set if (VEA) = 0, otherwise cleared
N Set equal to bit 0 of VEA
C Set equal to bit b of VEA
O Cleared

TSBM*,b,x A Test and Set Bit in Memory

0 7	8 11	12	13 15
CC	b	I	Rxx
Virtual Operand Address			

(VEA) → Result
1 → (VEA)b

TSBM is designed for multiple CPU intercommunication by semaphore. The contents of memory location VEA are tested and then bit b in memory location VEA is set to 1. During execution of this instruction the memory lock-out feature is used to prevent another user from accessing location VEA until after the bit has been set.

Protection of any semaphore location from other users within the computer, including I/O, is software dependent.

Affected: (VEA)b

Condition Codes:
N ~~Z~~ Set equal to bit 0 of stored VEA
Z ~~N~~ Reset
O ~~C~~ Reset
C ~~O~~ Set equal to bit b of result

TBMB*,b,x A,B Test Bit in Memory and Branch If One

0 7	8 11	12	13 15
86	b	I	Rxx
Virtual Operand Address			
Virtual Branch Address			

(VEA) → Result
If $(VEA)_b = 1$, ((PR)+2) → (PR)
If $(VEA)_b = 0$, (PR)+3 → (PR)

The contents of memory location VEA are tested. If the value in bit position b of memory location VEA is equal to one, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected None

Condition Codes:
Z Set if (VEA) = 0, otherwise cleared
N Set equal to bit 0 of VEA
C Set equal to bit b of VEA
O Cleared

TBSM,b,d — Test Bit(s) in Memory (Short-Displaced)

0 7	8 11	12 15
93	b	D

((R1)+D) → Result

The contents of short-displaced memory location VEA are tested.

Affected: None

Condition Codes:
- Z Set if (VEA) = 0, otherwise cleared
- N Set equal to bit 0 of VEA
- C Set equal to bit b of VEA
- O Cleared

TBSB,b,d B — Test Bit(s) in Memory (Short-Displaced) and Branch if One

0 7	8 11	12 15
96	b	D
Virtual Branch Address		

((R1)+D) → Result
If $((R1)+D)_b = 1$, ((PR)+1) → (PR)
If $((R1)+D)_b = 0$, (PR)+2 → (PR)

The contents of short-displaced memory location VEA are tested. If the value in bit position b of memory location VEA is equal to one, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes:
- Z Set if (VEA) = 0, otherwise cleared
- N Set equal to bit 0 of VEA
- C Set equal to bit b of VEA
- O Cleared

TBXM,b,x — Test Bit(s) in Memory (Short-Indexed)

0 7	8 11	12 15
9B	b	Rx

((Rx)) → Result

The contents of short-indexed memory location VEA are tested.

Affected: None

Condition Codes:
- Z Set if (VEA) = 0, otherwise cleared
- N Set equal to bit 0 of VEA
- C Set equal to bit b of VEA
- O Cleared

TBXB,b,x B — Test Bit(s) in Memory (Short-Indexed) and Branch if One

0 7	8 11	12 15
9E	b	Rx
Virtual Branch Address		

((Rx)) → Result
If $((Rx))_b= 1$,((PR)+1 → (PR)
If $((Rx))_b=0$,(PR)+2 → (PR)

The contents of short-indexed memory location VEA are tested. If the value in bit position b of memory location VEA is equal to one, a branch is executed to the virtual branch address; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes:
- Z Set if (VEA) = 0, otherwise cleared
- N Set equal to bit 0 of VEA
- C Set equal to bit b of VEA
- O Cleared

Branch and Hop Instructions

This instruction class provides the capability to perform both conditional and unconditional program branching, optional linking of the return address, and hopping relative to the current Program Counter. Conditional branch and hop operations always are based on testing the current status of one or more of the Condition Codes.

The only operands involved in any of these operations are word branch addresses, plus the linked return addresses which automatically are saved in a specified general purpose register.

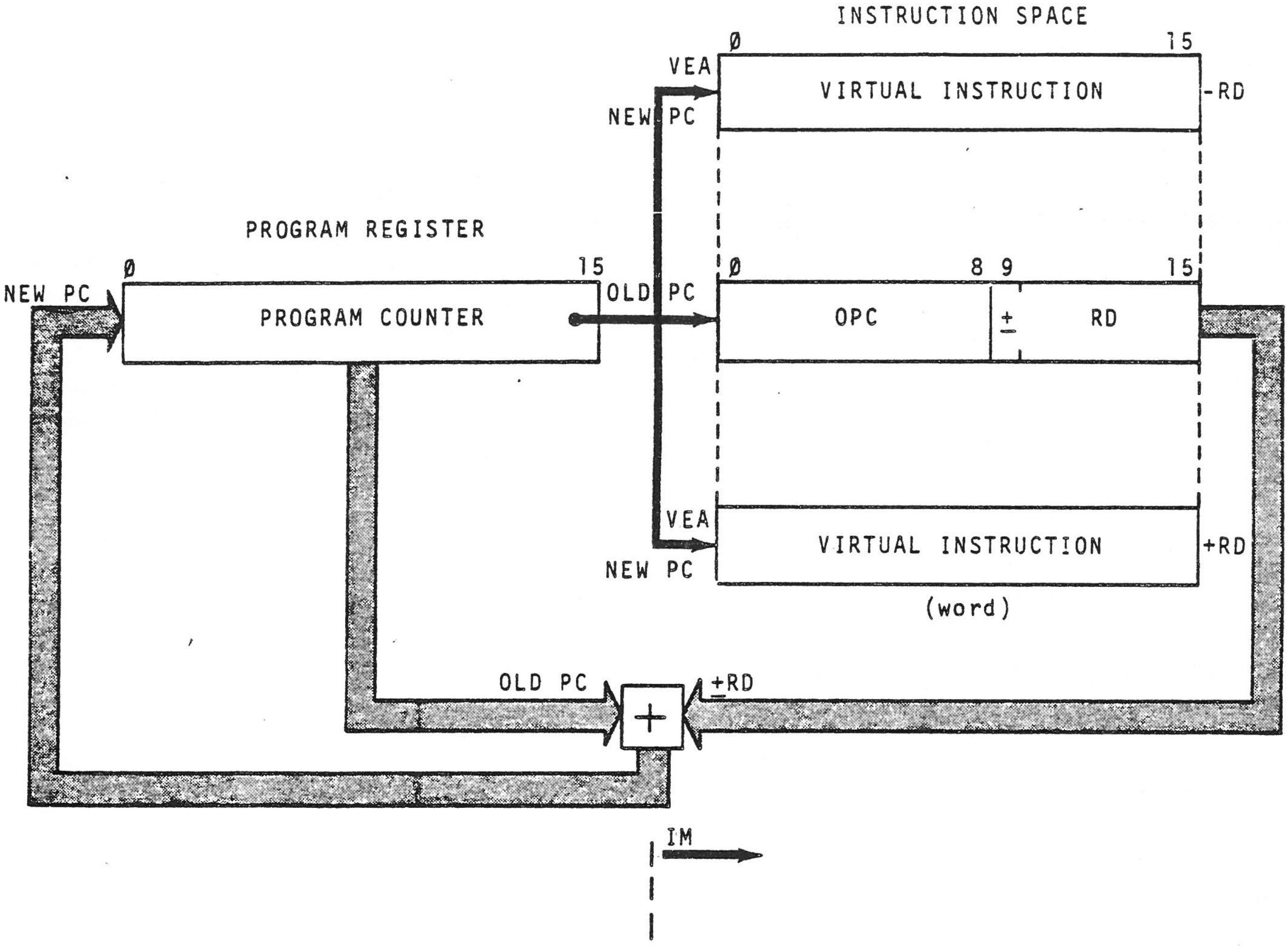
The branch operations primarily use the short-indexed addressing mode. Most branch and link operations use the direct addressing mode. All of the program hop operations specify a "relative displacement" of -64 to +63 memory locations from the current Program Counter for branching. Each time a branch or hop is executed, all 16 bits of the Program register are replaced and the pipeline is cleared. None of the instructions in this class set Condition Codes.

The specific meaning of the Condition Codes are defined by each instruction that affects them. Specific responses to those meanings are separately defined by the conditional branch (and link) and hop instructions that test them. The usual meanings of the four Condition Code bits are defined as follows:

Condition Code	*Bit Position in PSW*	*Usual Meaning*
N (CCN)	12	Result < 0
Z (CCZ)	13	Result = 0
O (CCO)	14	Overflow
C (CCC)	15	Carry Out

Relative-Displaced Addressing is used by all hop instructions to transfer program execution control relative to the instruction location. Execution control may be transferred wihtin a range from 64 memory locations below to 63 memory locations above the current instruction location in the virtual addressing space of the program's Instruction Map (IM).

All hop instructions consist of one memory word mapped in the virtual addressing space of the program's Instruction Map (IM). In addition to the operation code, the hop instruction word specifies the relative displacement (Rd) of the next instruction to be executed in the same virtual addressing space. This relative-displaced mode of addressing is diagrammed as follows:



Bits 9-15 (± RD in signed two's complement format) of the current instruction are added to the contents of the Program Register (old PC) to form the VEA (new PC) of the next instruction to execute in the program's virtual instruction addressing space (IM).

In addition to the branch and hop instructions, many instructions described in the preceeding sections provide the capability to branch unconditionally or conditionally on the result of an operation being nonzero, the result of a compare operation being equal or less, and the result of a test operation detecting any one's. Those instructions and their page numbers are referenced in the appropriate subsections that follow.

Branch Unconditionally

The following instructions cause a branch (or hop) to occur unconditionally to a specified memory location.

Three unconditional branch instructions described previously are cross-referenced as follows for convenience.

LOAD	**Bit(s) and Branch Unconditionally**	**Page**
LBRB	Load Bit in Register and Branch Unconditionally	9-4
GMRB	Generate Mask in Register and Branch Unconditionally	9-4

OR Bit and Branch Unconditionally		**Page**
OBRB	OR Bit in Register and Branch Unconditionally	9-69

The following additional instructions perform the unconditional branch functions exclusively.

BRU*,x B Branch Unconditionally

0 … 7	*8 … 11*	*12*	*… 15*
E7	0	I	Rxx
Virtual Branch Address			

VEA ⟶(PR)

A branch is executed to memory location VEA.

Affected: None

Condition Codes: Unaffected

BRX,x Branch (Short-Indexed) Unconditionally

0 … 7	*8 … 11*	*12 … 15*
FF	0	Rx

(Rx) ⟶(PR)

A branch is executed to short-indexed memory location VEA.

Affected: None

Condition Codes: Unaffected

HOP,d HOP Unconditionally

0 … 7	*8*	*… 15*
F7	0	± RD

(PR)+RD ⟶(PR)

A branch (hop) is executed to relative-displaced memory location VEA.

Affected: None

Condition Codes: Unaffected

Branch Conditionally

The following instructions cause a branch (or hop) to occur conditionally, either as a function of the result of instruction execution (computation), or based exclusively on the current state of one or more Condition Codes. If a branch (or hop) occurs, the pipeline will be cleared.

When Condition Codes are tested to determine branching, they are those resulting from the last operation to affect the Condition Codes. This group of conditional branch instructions are of three types:

1. branch on explicit status conditions of zero (Z), negative (N), carry (C), or overflow (O).
2. branch on implicit arithmetic comparative conditions between two operands.
3. branch on implicit unsigned magnitude comparative conditions between two operands.

Branching on explicit status conditions assumes the meaning of the last instruction to alter the Condition Codes. Branching on arithmetic conditions assumes the last instruction to alter the Condition Codes operated on fixed point or floating point signed quantities. Branching on magnitude conditions assumes preceding unsigned logical quantities.

Many instructions described previously branch unconditionally on the nonzero result of an operation, on the equal or less result of the arithmetic comparison of two operands, or on the one(s) result of the logical test (AND function) of two operands. These instructions are cross-referenced as follows for convenience:

Transfer And Branch if NonZero (ABINZ)

		Page
TRRB	Transfer Register to Register ABINZ	9-5.1

ADD And Branch if NonZero

ABRB	Add Bit in Register ABINZ	9-19
ADRB	Add Register to Register ABINZ	9-19
ABMB	Add Bit in Memory ABINZ	9-22
ABSB	Add Bit in Memory (Short-Displaced) ABINZ	9-23
ABXB	Add Bit in Memory (Short-Indexed) ABINZ	9-23
ADMB	Add Register to Memory ABINZ	9-24
ADSB	Add Register to Memory (Short-Displaced) ABINZ	9-24
ADXB	Add Register to Memory (Short-Indexed) ABINZ	9-25

SUBTRACT And Branch if NonZero

SBRB	Subtract Bit in Register ABINZ	9-26
SURB	Subtract Register from Register ABINZ	9-27

TWO'S COMPLEMENT and Branch if NonZero

		Page
TTRB	Transfer Two's Complement of Register to Register ABINZ	9-36

EXTRACT And Branch if NonZero

ETRB	Extract Register from Register ABINZ	9-65
ETMB	Extract Register from Memory ABINZ	9-67
ETSB	Extract Register from Memory (Short-Displaced) ABINZ	9-68
ETXB	Extract Register from Memory (Short-Indexed) ABINZ	9-68

OR And Branch If NonZero

ORRB	OR Register and Register ABINZ	9-70

EXCLUSIVE OR And Branch If NonZero

XBRB	Exclusive OR Bit in Register ABINZ	9-74
XORB	Exclusive OR Register and Register ABINZ	9-75

ZERO And Branch If NonZero

ZBRB	Zero Bit in Register ABINZ	9-77
ZBMB	Zero Bit in Memory ABINZ	9-77
ZBSB	Zero Bit in Memory (Short-Displaced) ABINZ	9-77
ZBXB	Zero Bit in Memory (Short-Indexed) ABINZ	9-78

COMPARE And Branch If Equal or Less (ABIEL)

CRMB	Compare Register to Memory ABIEL	9-91
CRSB	Compare Register to Memory (Short-Displaced) ABIEL	9-92
CRXB	Compare Register to Memory (Short-Indexed) ABIEL	9-93
CBMB	Compare Bit to Memory ABIEL	9-95
CBSB	Compare Bit to Memory (Short-Displaced) ABIEL	9-95
CBXB	Compare Bit to Memory (Short-Indexed) ABIEL	9-96

TEST And Branch If One(s) (ABIO)

TBRB	Test Bit in Register ABIO	9-97
TERB	Test Register and Register ABIO	9-98
TRMB	Test Register and Memory ABIO	9-98
TRSB	Test Register and Memory (Short-Displaced) ABIO	9-98
TRXB	Test Register and Memory (Short-Indexed) ABIO	9-98
TBMB	Test Bit in Memory ABIO	9-99

TBSB	Test Bit in Memory (Short-Displaced) ABIO	9-100
TBXB	Test Bit in Memory (Short-Indexed) ABIO	9-100

HNS,d Hop On Condition Code N Set

0 7	8	15
A8	9	±RD

If CCN = 1, (PR)+RD →(PR)
If CCN = 1, (PR)+1 →(PR)

If Condition Code N is set, a hop occurs to relative-displaced memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes: Tested only

HNR,d Hop On Condition Code N Reset

0 7	8	9 15
A8	1	±RD

If CCN = 0, (PR)+RD →(PR)
If CCN = 1, (PR)+1 →(PR)

If Condition Code N is reset, a hop occurs to relative-displaced memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes: Tested only

The following instructions perform the conditional branch or hop function exclusively. References are included as appropriate to the separate conditional branch and link instructions which branch only when link register Rd is specified to be R0.

HZS,d Hop On Condition Code Z Set

0 7	8	9 15
A9	0	±RD

If CCZ = 1, (PR)+RD →(PR)
If CCZ = 0, (PR)+1 →(PR)

If Condition Code Z is set a hop occurs to relative-displaced memory location VEA; otherwise the next instruction in sequence is executed.

Affected: None

Condition Codes: Tested only

HZR,d Hop On Condition Code Z Reset

0 7	8	9 15
A9	1	±RD

If CCZ = 0, (PR)+RD →(PR)
If CCZ = 1, (PR)+1 →(PR)

If Condition Code Z is reset a hop occurs to relative-displaced memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes: Tested only

HOS,d Hop On Condition Code O Set

0 7	8	9 15
AA	0	+ - RD

If CCO = 1, (PR)+RD → (PR)
If CCO = 0, (PR)+1 → (PR)

If Condition Code O is set, a hop occurs to relative-displaced memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition
Codes: Tested only

HOR,d Hop on Condition Code O Reset

0 7	8	9 15
AA	1	+ - RD

If CCO = 0, (PR)+RD → (PR)
If CCO = 1, (PR)+1 → (PR)

If Condition Code O is reset, a hop occurs to relative-displaced memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition
Codes: Tested only

HCS,d Hop On Condition Code C Set

0 7	8	9 15
AB	0	+ - RD

If CCC = 1, (PR)+RD → (PR)
If CCC = 0, (PR)+1 → (PR)

If Condition Code C is set, a hop occurs to relative displaced memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition
Codes: Tested only

HCR,d Hop On Condition Code C Reset

0 7	8	9 15
AB	1	+ - Rd

If CCC = 0, (PR)+RD → (PR)
If CCC = 1, (PR)+1 → (PR)

If Condition Code C is reset, a hop occurs to relative-displaced memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition
Codes: Tested only

HLS,d — Hop on Less Than Condition

0 7	8	9 15
AC	0	+- RD

CCN .XOR. CCO → Result
If CCN .XOR. CCO = 1, (PR)+RD → (PR)
If CCN .XOR. CCO = 0, (PR)+1 → (PR)

If only one of the negative or overflow conditions are set, the preceding instruction's first (destination) operand is algebraically less than the second (source) operand and a hop is executed to relative-displaced memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes: Tested only

HGE,d — Hop on Greater Than or Equal Condition

0 7	8	9 15
AC	1	+- RD

CCN .XOR. CCO → Result
If CCZ V ($\overline{\text{CCZ}}$ .AND. ($\overline{\text{CCN}}$ .XOR. CCO)) = 1, (PR)+RD → (PR)
If CCZ V ($\overline{\text{CCZ}}$ .AND. ($\overline{\text{CCN}}$ .XOR. CCO)) = 0, (PR)+1 → (PR)

If the zero condition is set, or zero is not set and neither or both of the negative and overflow conditions are set the preceding instruction's first (destination) operand is algebraically greater than or equal to the second (source) operand and a hop is executed to a relative-displaced memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes: Tested only

HGT,d — Hop on Greater Than Condition

0 7	8	9 15
AD	1	+- RD

CCZ V (CCN .XOR. CCO) → Result
If CCZ V (CCN .XOR. CCO) = 0, (PR)+RD → (PR)
If CCZ V (CCN .XOR. CCO) = 1, (PR)+1 → (PR)

If the zero condition is not set and both or neither of the negative or overflow conditions are set, the preceding instruction's first (destination) operand is algebraically greater than the second (source) operand and a hop is executed to relative-displaced memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes: Tested only

HLE,d — Hop on Less Than or Equal Condition

0 7	8	9 15
AD	0	+- RD

CCZ V (CCN .XOR. C$\overline{\text{CO}}$ → Result
If CCZ V (CCN .XOR. CCO) = 1, (PR)+RD → (PR)
If CCZ V (CCN .XOR. CCO) = 0, (PR)+1 → (PR)

Unless the zero condition is not set and both or neither of the negative or overflow conditions are set, the preceding instruction's first (destination) operand is algebraically less than or equal to the second (source) operand and a hop is executed to relative-displaced memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes: Tested only

HHI,d — Hop on Magnitude Higher Condition

0 7	8	9 15
A6	0	+ - RD

$\overline{\text{CCC}}$ V CCZ → Result
If $\overline{\text{CCC}}$ V CCZ=0, (PR)+RD → (PR)
If $\overline{\text{CCC}}$ V CCZ=1,(PR)+1 → (PR)

If the carry condition is set, and the zero condition is reset, the preceding instruction's first (destination) operand is higher in magnitude than the second (source) operand and a hop is executed to relative-displaced memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes: Tested only

HNH,d — Hop on Magnitude Not Higher Condition

0 7	8	9 15
A6	1	+ - RD

$\overline{\text{CCC}}$ V CCZ → Result
If $\overline{\text{CCC}}$ V CCZ = 1, (PR)+RD → (PR)
If $\overline{\text{CCC}}$ V CCZ = 0, (PR)+1 → (PR)

If the carry condition is reset, and/or the zero condition is set, the preceding instruction's first (destination) operand is not higher in magnitude than the second (source) operand and a hop is executed to relative-displaced memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes: Tested only

BXNS,x — Branch (Short-Indexed) on Condition Code N Set

0 7	8 11	12 15
8F	#0	Rx

If CCN = 1, (Rx) → (PR)
If CCN = 0, (PR)+1 → (PR)

If Condition Code N is set, a branch is executed to short-indexed memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes: Tested only

BXNR,x — Branch (Short-Indexed) On Condition Code N Reset

0 7	8 11	12 15
8F	#8	Rx

If CCN = 0, (Rx) → (PR)
If CCN = 1, (PR)+1 → (PR)

If Condition Code N is reset a branch is executed to short-indexed memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes: Tested only

BXZS,x Branch (Short-Indexed) on Condition Code Z Set

0 7	8 11	12 15
8F	#1	Rx

If CCZ = 1, (Rx) → (PR)
If CCZ = 0, (PR)+1 → (PR)

If Condition Code Z is set, a branch is executed to short-indexed memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes: Tested only

BXZR,x Branch (Short-Indexed)on Condition Code Z Reset

0 7	8 11	12 15
8F	#9	Rx

If CCZ = 0, (Rx) → (PR)
If CCZ = 1, (PR)+1 → (PR)

If Condition Code Z is reset a branch is executed to short-indexed memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes: Tested only

BXOS,x Branch (Short-Indexed)on Condition Code O Set

0 7	8 11	12 15
8F	#2	Rx

If CCO = 1, (Rx) → (PR)
If CCO = 0, (PR)+1 → (PR)

If Condition Code O is set, a branch is executed to short-indexed memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes: Tested only

BXOR,x Branch (Short-Indexed) on Condition Code O Reset

0 7	8 11	12 15
8F	#A	Rx

If CCO = 0, (Rx) → (PR)
If CCO = 1, (PR)+1 → (PR)

If Condition Code O is reset, a branch is executed to short-indexed memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes: Tested only

BXCS,x Branch (Short-Indexed) on Condition Code C Set

0 7	8 11	12 15
8F	#3	Rx

If CCC = 1, (Rx) →(PR)
If CCC = 0, (PR)+1 →(PR)

If Condition Code C is set, a branch is executed to short-indexed memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition
Codes: Tested only

BXCR,x Branch (Short-Indexed) on Condition Code C Reset

0 7	8 11	12 15
8F	#B	Rx

If CCC = 0, (Rx) →(PR)
If CCC = 1, (PR)+1 →(PR)

If Condition Code C is reset, a branch is executed to short-indexed memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition
Codes: Tested only

BXLS,x Branch (Short-Indexed) on Less Than Condition

0 7	8 11	12 1 15
8F	#4	Rx

CCN .XOR. CCO → Result
If CCN .XOR. CCO = 1, (Rx) →(PR)
If CCN .XOR. CCO = 0, (PR)+1 →(PR)

If only one of the negative or overflow conditions is set, the preceding instruction's first (destination) operand is algebraically less than the second (source) operand and a branch is executed to short-indexed memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition
Codes: Tested only

BXGE,x Branch (Short-Indexed) on Greater Than or Equal Condition

0 7	8 11	12 15
8F	#C	Rx

If CCZ V ($\overline{\text{CCZ}}$ .AND. ($\overline{\text{CCN}}$ .XOR. CCO)) = 1, (Rx) →(PR)
If CCZ V ($\overline{\text{CCZ}}$ .AND. ($\overline{\text{CCN}}$ .XOR.CCO)) = 0, (PR)+1 →(PR)

If the zero condition is set, or zero is not set and neither or both of the negative and overflow conditions are set the preceding instruction's first (destination) operand is algebraically greater than or equal to the second (source) operand and a branch is executed to short-indexed memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition
Codes: Tested only

BXGT,x Branch (Short-Indexed) on Greater Than Condition

0 7	8 11	12 15
8F	#D	Rx

CCZ V (CCN .XOR. CCO) → Result
If CCZ V (CCN .XOR. CCO) = 0, (Rx) → (PR)
If CCZ V (CCN .XOR. CCO) = 1, (PR)+1 → (PR)

If the zero condition is not set and both or neither of the negative or overflow conditions are set, the preceding instruction's first (destination) operand is algebraically greater than the second (source) operand and a branch is executed to short-indexed memory location VEA; otherwise the next instruction in sequence is executed.

Affected: None

Condition Codes: Tested only

BXLE,x Branch (Short-Indexed) on Less Than or Equal Condition

0 7	8 11	12 15
8F	#5	Rx

CCZ V (CCN .XOR. CCO) → Result
If CCZ V (CCN .XOR. CCO) = 1 (Rx) → (PR)
If CCZ V (CCN .XOR. CCO) = 0 (PR)+1 → (PR)

Unless the zero condition is not set and both or neither of the negative or overflow conditions are set, the preceding instruction's first (destination) operand is algebraically less than or equal to the second (source) operand and a branch is executed to short-indexed memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes: Tested only

BXHI,x Branch (Short-Indexed) on Magnitude Higher Condition

0 7	8 11	12 15
8F	#6	Rx

$\overline{CCC}$ V CCZ → Result
If $\overline{CCC}$ V CCZ = 0, (Rx) → (PR)
If $\overline{CCC}$ V CCZ = 1, (PR)+1 → (PR)

If the carry condition is set, and the zero condition is reset, the preceding instruction's first (destination) operand is higher in magnitude than the second (source) operand and a branch is executed to short-indexed memory location VEA; otherwise the next instruction in sequence is executed.

Affected: None

Condition Codes: Tested only

BXNH,x Branch (Short-Indexed) on Magnitude Not Higher Condition

0 7	8 11	12 15
8F	#E	Rx

$\overline{CCC}$ V CCZ → Result
If $\overline{CCC}$ V CCZ = 1, (Rx) → (PR)
If $\overline{CCC}$ V CCZ = 0, (PR)+1 → (PR)

If the carry condition is reset or the zero condition is set, the preceding instruction's first (destination) operand is not higher in magnitude than the second (source) operand and a branch is executed to short-indexed memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: None

Condition Codes: Tested only

Branch and Link Unconditionally

The following instructions cause a branch to occur unconditionally to a specified memory location. At the same time, the location of the next instruction in sequence following the current instruction is saved in the specified general purpose register. The pipeline will also be cleared.

BLM*,r,x B — Branch and Link

0 7	8 11	12	13 15
E7	Rd	I	Rxx
Virtual Branch Address			

(PR)+2 → (Rd)
VEA → (PR)

The contents of the Program Register plus two replace the contents of register Rd and a branch is executed to memory location VEA.

Affected: (Rd)

Condition Codes: Unaffected

BLI,r,~~x~~ — Branch and Link (Immediate)

0 7	8 11	12 15
EF	Rd	0

(PR)+2 → (Rd)
(PR)+1 → (PR)

The contents of the Program Register plus two replace the contents of register Rd and the next instruction in sequence is executed.

Affected: (Rd)

Condition Codes: Unaffected

BLX,r,x — Branch and Link (Short-Indexed)

0 7	8 11	12 15
FF	Rd not 0	Rx

(PR)+1 → (Rd)
(Rx) → (PR)

The contents of the Program Register plus one replace the contents of register Rd and a branch is executed to short-indexed memory location VEA. (For Rd=0 see the BRX instruction.)

Affected: (Rd)

Condition Codes: Unaffected

BLT,d — Branch and Link (Indexed-Through-Table)

0 7	8 11	12 15
F7	1	+ - RD

(PR)+1 → (R8)
((R2)+RD) → (PR)

The contents of the Program Register plus one replace the contents of register R8. A branch is executed to effective memory location VEA in the program's virtual instruction space. Location VEA is contained in relative-displaced position RD of a table in the program's virtual operand space whose base address is contained in register R2.

Affected: (R8)

Condition Codes: Unaffected

Branch and Link Conditionally

The following instructions cause a branch to occur conditionally, based on the current state of one or more Condition Codes, to a direct addressed memory location. If the branch is executed, the location of the next instruction in sequence following the branch instruction is saved in a specified general purpose register. If register zero is specified, the return address is not saved and the instruction executes only as a conditional branch operation. All branches will clear the pipeline.

BLNS,r B — Branch and Link on Condition Code N Set

0 7	8 11	12 15
A7	#0	Rd
Virtual Branch Address		

If CCN = 1, (PR)+2 → (Rd) and VEA → (PR)
If CCN = 0, (PR)+2 → (PR)

If Condition code N is set, the contents of the Program Register plus two replace the contents of register Rd (if Rd ≠ 0) and a branch is executed to direct addressed memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: (Rd) if branch, otherwise none

Condition Codes: Tested only

BLNR,r B — Branch and Link on Condition Code N Reset

0 7	8 11	12 15
A7	#8	Rd
Virtual Branch Address		

If CCN = 0, (PR)+2 → (Rd) and VEA → (PR)
If CCN = 1, (PR)+2 → (PR)

If Condition code N is reset, the contents of the Program Register plus two replace the contents of register Rd (if Rd ≠ 0) and a branch is executed to direct addressed memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: (Rd) if branch, otherwise none

Condition Codes: Tested only

BLZS,r B — Branch and Link on Condition Code Z Set

0 7	8 11	12 15
A7	#1	Rd
Virtual Branch Address		

If CCZ = 1, (PR)+2 → (Rd) and VEA → (PR)
If CCZ = 0, (PR)+2 → (PR)

If Condition Code Z is set, the contents of the Program Register plus two replace the contents of register Rd (if Rd ≠ 0) and a branch is executed to direct addressed memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: (Rd) if branch, otherwise none

Condition Codes: Tested only

BLZR,r B — Branch and Link on Condition Code Z Reset

0 7	8 11	12 15
A7	#9	Rd
Virtual Branch Address		

If CCZ = 0, (PR)+2 → (Rd) and VEA → (PR)
If CCZ = 1, (PR)+2 → (PR)

If Condition Code Z is set, the contents of the Program Register plus two replace the contents of register Rd (if Rd ≠ 0) and a branch is executed to direct addressed memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: (Rd) if branch, otherwise none

Condition Codes: Tested only

BLOS,r B — Branch and Link on Condition Code O Set

0 7	8 11	12 15
A7.	#2	Rd
Virtual Branch Address		

If CCO = 1,(PR)+2 → (Rd) and VEA → (PR)
If CCO = 0, (PR)+2 → (PR)

If Condition Code 0 is set, the contents of the Program Register plus two replace the contents of register Rd (if Rd ≠ 0) and a branch is executed to direct addressed memory location VEA; otherwise the next instruction in sequence is executed.

Affected: (r (Rd) if branch, otherwise none

Condition Codes: Tested only

BLOR,r B — Branch and Link on Condition Code O Reset

0 7	8 11	12 15
A7	#A	Rd
Virtual Branch Address		

If CCC = 0, (PR)+2 → (Rd) and VEA → (PR)
If CCC = 0, (PR)+2 → (PR)

If Condition Code 0 is reset, the contents of the Program Register plus two replace the contents of register Rd (if Rd ≠ 0), and a branch is executed to direct addressed memory location VEA; otherwise the next instruction in sequence is executed.

Affected: (Rd) if branch, otherwise none

Condition Codes: Tested only

BLCS,r B Branch and Link on Condition Code C Set

0 7	8 11	12 15
A7	#3	Rd
Virtual Branch Address		

If CCC = 1, (PR)+2 →(Rd) and VEA →(PR)
If CCC = 0, (PR)+2 →(PR)

If Condition Code C is set, the contents of the Program Register plus two replace the contents of register Rd (if Rd ≠ 0) and a branch is executed to direct addressed memory location VEA; otherwise the next instruction in sequence is executed.

Affected: (Rd) if branch, otherwise none

Condition Codes: Tested only

BLCR,r B Branch and Link on Condition Code C Reset

0 7	8 11	12 15
A7	#B	Rd
Virtual Branch Address		

If CCO = 0, (PR)+2 →(Rd) and VEA →(PR)
If CCC = 1, (PR)+2 →(PR)

If Condition Code C is reset, the contents of the Program Register plus two replace the contents of register Rd (if Rd ≠ 0) and a branch is executed to direct addressed memory location VEA; otherwise,the next instruction in sequence is executed.

Affected: (Rd) if branch, otherwise none

Condition Codes: Tested only

BLLS,r B Branch and Link on Less Than Condition

0 7	8 11	12 15
A7	#4	Rd
Virtual Branch Address		

CCN .XOR. CCO → Result
If CCN .XOR. CCO = 1,(PR)+2 →(Rd) and VEA →(PR)
If CCN .XOR. CCO = 0, (PR)+2 →(PR)

If only one of the negative and overflow conditions is set, the preceding instruction's first (destination) operand is algebraically less than the second (source) operand. As a result, the contents of the Program Register plus two replace the contents of register Rd (if Rd ≠ 0) and a branch is executed to direct addressed memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: (Rd) if branch, otherwise none

Condition Codes: Tested only

BLGE,r B Branch and Link on Greater Than or Equal Condition

0 7	8 11	12 15
A7	#C	Rd
Virtual Branch Address		

If CCZ V ($\overline{\text{CCZ}}$ .AND. ($\overline{\text{CCN}}$ .XOR. CCO)) = 1, (PR)+2 →(Rd)
and VEA →(PR)
If CCZ V ($\overline{\text{CCZ}}$ .AND. ($\overline{\text{CCN}}$ .XOR. CCO)) = 0, (PR)+2 →(PR)

If the zero condition is set, or zero is not set and neither or both of the negative and overflow conditions are set the preceding instruction's first (destination) operand is algebraically greater than or equal to the second (source) operand. As a result, the contents of the Program Register plus two replace the contents of register Rd (if Rd ≠ 0) and a branch is executed to direct addressed memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: (Rd) if branch, otherwise none

Condition Codes: Tested only

BLGT,r B — Branch and Link on Greater Than Condition

0 7	8 11	12 15
A7	#D	Rd
Virtual Branch Address		

CCZ V (CCN .XOR. CCO) → Result

If CCZ V (CCN .XOR. CCO) = 0, (PR)+2 → (Rd) and VEA → (PR)

If CCZ V (CCN .XOR. CCO) = 1, (PR)+2 → (PR)

If the zero condition is not set and both or neither of the negative or overflow conditions are set, the preceding instruction's first (destination) operand is algebraically greater than the second (source) operand. As a result, the contents of the Program Register plus two replace the contents of register Rd (if Rd ≠ 0) and a branch is executed to direct addressed memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: (Rd) if branch, otherwise none

Condition
Codes Tested only

BLLE,r B — Branch and Link on Less Than or Equal Condition

0 7	8 11	12 15
A7	#5	Rd
Virtual Branch Address		

CCZ V (CCN .XOR. CCO) → Result

If CCZ V (CCN .XOR. CCO) = 1, (PR)+2 → (Rd) and VEA → (PR)

If CCZ V (CCN .XOR. CCO) = 0, (PR)+2 → (PR)

Unless the zero condition is not set and both or neither of the negative or overflow conditions are set, the preceding instruction's first (destination) operand is algebraically less than or equal to the second (source) operand. As a result, the contents of the Program Register plus two replace the contents of register Rd (if Rd ≠ 0) and a branch is executed to direct addressed memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: (Rd) if branch, otherwise none

Condition
Codes: Tested only

BLHI,r B — Branch and Link on Magnitude Higher Condition

0 7	8 11	12 15
A7	#6	Rd
Virtual Branch Address		

$\overline{\text{CCC}}$ V CCZ → Result

If $\overline{\text{CCC}}$ V CCZ = 0, (PR)+2 → (Rd) and VEA → (PR)

If $\overline{\text{CCC}}$ V CCZ = 1, (PR)+2 → (PR)

If the carry condition is set and the zero condition is reset, the preceding instruction's first (destination) operand is higher in magnitude than the second (source) operand. As a result, the contents of the Program Register plus two replace the contents of register Rd (if Rd ≠ 0) and a branch is executed to direct addressed memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: (Rd) if branch, otherwise none

Condition
Codes: Not tested

BLNH,r B — Branch and Link on Magnitude Not Higher Condition

0 7	8 11	12 15
A7	#E	Rd
Virtual Branch Address		

$\overline{\text{CCC}}$ V CCZ → Result

If $\overline{\text{CCC}}$ V CCZ = 1, (PR)+2 → (Rd) and VEA → (PR)

If $\overline{\text{CCC}}$ V CCZ = 0, (PR)+2 → (PR)

If the carry condition is reset or the zero condition is set, the preceding instruction's first (destination) operand is not higher in magnitude than the second (source) operand. As a result, the contents of the Program Register plus two replace the contents of register Rd (if Rd ≠ 0) and a branch is executed to direct addressed memory location VEA; otherwise, the next instruction in sequence is executed.

Affected: (Rd) if branch, otherwise none

Condition
Codes: Not tested

Control Instructions

This instruction class provides the capability to perform both privileged and non-privileged Central Processor and Memory System control functions. Many of the instructions in this class use special memory, and CPU control register(s) addressing modes. Some of these instructions also set Condition Codes.

Following is a list of the separate non-privileged and privileged control instructions.

Non-Privileged Control Instruction

NOP	No Operation
EXR	Execute Instruction Word in Register(s)
EXI	Execute Immediate
TRO	Transfer and Reset Overflow Status History in PSD
LCCC	Load Register with Current Condition Codes of PSD
LCPS	Load Register with Current Program Status Register of PSD
LCPR	Load Register with Current Program Register of PSD
SCCC	Select Current Condition Codes in PSD
LTDL	Loop Termination, Direct Control Variable, Literal Limit
LTDD	Loop Termination, Direct Control Variable, Direct Limit
LTIL	Loop Termination, Indirect Control Variable, Literal Limit
LTID	Loop Termination, Indirect Control Variable, Direct Limit
EXMA	Enter Extended Memory Addressing
REX	Request Executive Service
LCIE	Load Register with Current Interrupt Enable Latches
LCIR	Load Register with Current Interrupt Request Latches
LCIA	Load Register with Current Interrupt Active Latches

Privileged Control Instructions

HLT	Halt
EVMO	Enter Virtual Mode of CPU Execution
XVMO	Exit Virtual Mode of CPU Execution
EPMD	Enter Pipelined Mode of Execution
XPMD	Exit Pipelined Mode of Execution
MRBM	Move Register Block Section of Context File to Memory File
MMRB	Move Memory File to Register Block Section of Context File
SCRB	Select Current Register Block in PSD
LIMP	Load Instruction Map Image into Hardware Map
LOMP	Load Operand Map Image into Hardware Map
ZIMP	Zero Section of Instruction Map
ZOMP	Zero Section of Operand Map
SIOM	Select Another Program's IM as Current OM
SOOM	Select Another Program's OM as Current OM
SZOM	Select Map Zero as Current OM
LDVM	Load Register from Memory (Via Map Image)
STVM	Store Register into Memory (Via Map Image)
LDAM	Load Register from (Actual) Memory
STAM	Store Register into (Actual) Memory
RMPS	Read Memory Plane Status
RMWS	Read Memory Word Status
WMS	Write Memory Status
LXR	Load Extended Memory Control Register
BRM	Branch to Micro Location
BRMI	Branch to Micro Location Immediate
RDI	Read Internal Register
WRI	Write Internal Register
SIE	Set Interrupt Enable
SIR	Set Interrupt Request
SIA	Set Interrupt Active
RIE	Reset Interrupt Enable
RIR	Reset Interrupt Request
RIA	Reset Interrupt Active
CAR	Clear Active and Return
CIR	Clear Interrupt and Return
RMI	Request Multiprocessor Interrupt
ISA,B, C,D	Input Status from I/O Group A,B,C, or D
IDA,B, C,D	Input Data from I/O Group A,B,C, or D
OCA,B C,D	Output Command to I/O Group A,B,C, or D
ODA,B, C,D	Output Data to I/O Group A,B,C, or D
DMPI	Initialize Direct Memory Processor

No Operation

The following instructions are used to idle the CPU through one instruction without affecting the contents or status of any control elements.

NOP No Operation

0 7	8 11	12 15
F7	0	1

A non-operation occurs and the next instruction in sequence is executed. This instruction is implemented as a HOP to location PR + 1 which will contain the next sequential instruction. In contrast to the 8900 op code below, this instruction will result in the PR being reloaded instead of incremented, causing a CPU memory interface flush.

Affected: None

Condition Codes: Unaffected

NOP No Operation

0 7	8 11	12 15
89	0	0

A non-operation occurs and the next instruction in sequence is executed. In contrast to the F701 instruction above, this op code will not cause a memory interface flush because the PR is incremented, not reloaded. However, this op code will result in R0 being updated with the contents of the control panel switch register.

Affected: R0

Condition Codes: Unaffected

Execute Instruction Word in Register(s)

The following instructions are used to execute an instruction in the general purpose registers. The first word (or only word) of the instruction is formed from the logical OR of the contents of two separate registers or a register and memory location. Neither register is modified nor is the memory location. If the two registers are designated to be the same register, then the instruction in that register will be executed unmodified.

The register(s) instruction executes as through it were located at the same memory address as the EXR or EXI instruction. For multi-word register instructions, subsequent words must follow the EXR instruction word in the program's virtual instruction space (i.e., (PR)+1, (PR)+2 etc).

Condition Codes set are those of the executed register instruction. EXR and EXI are not privileged, but will cause a privileged instruction violation if the calling program is unprivileged and the executed register instruction is privileged.

EXR,r,s Execute Instruction Word in Register(s)

0 7	8 11	12 15
B3	Rd	Rs

(Rd)V(Rs) →(IR)

The logical sum (OR function) of the contents of register Rd and register Rs replace the contents of the instruction Register (IR). The instruction execution cycle continues with the replaced contents of IR. (*Execution time of resultant executed instruction.)

Affected: (IR)

Condition Codes: Those of the resultant executed instruction

EXI,r V Execute Immediate

0 7	8 11	12 15
E8	Rd	5
Immediate Operand		

(Rd) V ((PR)+1) →(IR)

The logical OR of the contents of the immediate memory location and register Rd replace the contents of the instruction register. The instruction execution cycle continues with the replaced contents of the IR.

Affected: IR

Condition Codes: Those of the resultant instruction

Load Register with Current Field(s) of PSD

The following instructions are used to load a general purpose register with either the current Integer Overflow History, Condition Codes, Program Status (PS) or the Program Register (PR) of the active Program Status Doubleword (PSD).

The transfer of current Integer Overflow History causes its status to be cleared in the active PSD and in the overflow Condition Code. Otherwise, the Condition Codes are unaffected by these instructions.

TRO,r — Transfer and Reset Overflow Status History in PSD

0 — 7	8 — 11	12 — 15
0E	Rd	0

$(PS)_{11} \longrightarrow (Rd)_0$
0's $\longrightarrow (Rd)_{1-15}$
0's $\longrightarrow (PS)_{11}$

The contents of bit 11 in the Program Status (word 1) in the active PSD are transferred to bit 0 of register Rd. Bits 1-15 of register Rd are set to zero. Bit 11 of the PS is reset to 0.

Affected: (Rd), $(PS)_{11}$

Condition Codes: Unaffected

LCCC,r — Load Register with Current Condition Codes of PSD

0 — 7	8 — 11	12 — 15
0E	Rd	3

$(PS)_{12-15} \longrightarrow (Rd)_{12-15}$
0's $\longrightarrow (Rd)_{0-11}$

The contents of bits 12-15 in the Program Status (word 1) in the active PSD are transferred to bits 12-15 of register Rd. Bits 0-11 of register Rd are set to zero.

Affected: (Rd)

Condition Codes: Unaffected

LCPS,r — Load Register with Current Program Status Register of PSD

0 — 7	8 — 11	12 — 15
0E	Rd	1

$(PS)_{0-15} \longrightarrow (Rd)$

The contents of the Program Status (word 1) in the active PSD are transferred to register Rd.

Affected: (Rd)

Condition Codes: Unaffected

LCPR,r — Load Register with Current Program Register of PSD

0 — 7	8 — 11	12 — 15
0E	Rd	2

$(PR)_{0-15} \longrightarrow (Rd)$

The contents of the Program Register (word 0) in the active PSD are transferred to register Rd.

Affected: (Rd)

Condition Codes: Unaffected

Select Current Condition Codes in PSD

The following instruction is used to set the four Condition Codes contained in a general purpose register into the current Program Status (PS) of the active Program Status Doubleword (PSD).

SCCC,s Select Current Condition Codes in PSD

0 7	8 11	12 15
B2	0	Rs

$(Rs)_{12\text{-}15} \rightarrow (PS)_{12\text{-}15}$

The contents of bits 12-15 in register Rs replace the contents of bits 12-15 in the Program Status (word 1) in the active PSD.

Affected: $(PS)_{12\text{-}15}$

Condition Codes: (Re)set to bits 12-15 of Rs

Halt Execution

The following instruction is used to halt program execution. Execution may be resumed by activation of the RUN/HALT control switch on the CPU Control Panel.

Execution of this instruction is permissable only in the Privileged State. The Condition Codes are unaffected.

HLT Halt (Privileged)

0 7	8 11	12 15
00	0	0

(Execution halts)

Program Execution is halted. The halt occurs after the Program Register is advanced and the next instruction is transferred to the Instruction Register.

Affected: None

Condition Codes: Unaffected

Loop Termination Via Control Variable

The following instructions are used to control the looping of program execution through some number of instructions by incrementing a variable either directly or indirectly addressed within the instruction. The incremented control value is then compared to a terminal value that is either given or directly addressed within the instruction.

As long as the terminal value is greater than the control variable, program execution branches to an address also given in the instruction; otherwise, execution continues with the next instruction. Condition Codes are affected as shown for each instruction in this grouping.

LTDL Loop Termination With Directly Addressed Control Variable, and Literal Terminal Value

0 7	8 11	12 15
E8	0	D
Terminal Value		
Control Variable Address		
Branch Address		

```
(((PR)+2))+1                          →(((PR)+2))
(((PR)+2)) - ((PR)+1)                 →Result
If (((PR)+2)) .LE. ((PR)+1), ((PR)+3) →(PR)
Otherwise, (PR)+4                     →(PR)
```

The contents of the memory location directly pointed to by the contents of (PR)+2 are incremented and stored in that location.

The contents of (PR)+1 are algebraically subtracted from the new contents of ((PR)+2) and the result is not stored. If the new contents of ((PR)+2 are less than or equal to ((PR)+1), a branch is executed to the location pointed to by ((PR)+3). Otherwise, the next instruction in sequence is executed.

Affected: 9 (((PR)+2))

Condition Codes:

- Z Set if (((PR)+2))=((PR)+1)
- N Set if (((PR+2)) < ((PR)+1)
- C Set if carry from most significant bit of result, otherwise cleared
- O Set if operands were of opposite sign and the sign of the subtrahend was the same as the sign of the result, otherwise cleared

LTDD Loop Termination With Directly Addressed Control Variable, and Directly Addressed Terminal Value

0 7	8 11	12 15
E8	0	F
Terminal Value Address		
Control Variable Address		
Branch Address		

```
                (((PR)+2))+1                  →(((PR)+2))
(((PR)+2)) - (((PR)+1))                       →Result
If (((PR)+2)) .LE. (((PR)+1)),((PR)+3)        →(PR)
Otherwise, (PR)+4                             →(PR)
```

The contents of the memory location directly pointed to by the contents of (PR)+2 are incremented and stored in that location. The contents of ((PR)+1) are algebraically subtracted from the contents of ((PR)+2), and the result is not stored. If the new contents of ((PR)+2) are less than or equal to (((PR)+1)), a branch is executed to the location pointed to by ((PR)+3). Otherwise, the next instruction in sequence is executed.

Affected: (((PR)+2))

Condition Codes:

- Z Set if (((PR)+2)) = ((PR)+1)
- N Set if (((PR+2)) < ((PR)+1)
- C Set if carry from most significant bit of result, otherwise cleared
- O Set if operands were of opposite sign and the sign of the subtrahend was the same as the sign of the result, otherwise cleared

LTIL Loop Termination With Indirectly Addressed Control Variable and Literal Terminal Value

<table>
<tr><th>0 … 7</th><th>8 … 11</th><th>12 … 15</th></tr>
<tr><td>E8</td><td>0</td><td>C</td></tr>
<tr><td colspan="3">Terminal Value</td></tr>
<tr><td colspan="3">Control Variable Address</td></tr>
<tr><td colspan="3">Branch Address</td></tr>
</table>

```
((((PR)+2)))+1                              → ((((PR)+2)))
((((PR))+2))) - ((PR)+1)                    → Result
If ((((PR)+2))) .LE. ((PR)+1),((PR)+3)      → (PR)
Otherwise, (PR)+4                           → (PR)
```

The contents of the memory location indirectly pointed to by the contents of (PR)+2 are incremented and stored in that location. The contents of (PR)+3 are algebraically subtracted from the contents of (((PR)+2)), and the result is not stored. If the new contents of (((PR)+2)) are less than or equal to ((PR)+1), a branch is executed to the location pointed to by ((PR)+3). Otherwise, the next instruction in sequence is executed.

Affected: ((((PR)+2)))

Condition Codes:

Z Set if (((PR)+2)) = ((PR)+1

N Set if (((PR+2)) < ((PR)+1)

C Set if carry from most significant bit of result, otherwise cleared

O Set if operands were of opposite sign and the sign of the subtrahend was the same as the sign of the result, otherwise cleared

LTID Loop Termination With Indirectly Addressed Control Variable and Directly Addressed Terminal Value

<table>
<tr><th>0 … 7</th><th>8 … 11</th><th>12 … 15</th></tr>
<tr><td>E8</td><td>0</td><td>E</td></tr>
<tr><td colspan="3">Terminal Value Address</td></tr>
<tr><td colspan="3">Control Variable Address</td></tr>
<tr><td colspan="3">Branch Address</td></tr>
</table>

```
((((PR)+2)))+1                              → ((((PR)+2)))
((((PR)+2))) - (((PR)+3))                   → Result
If ((((PR)+2))) .LE. (((PR)+1)),((PR)+3)    → (PR)
Otherwise, (PR)+4                           → (PR)
```

The contents of the memory location indirectly pointed to by the contents of (PR)+2 are incremented and stored in that location. The contents of ((PR)+1) are algebraically subtracted from the contents of (((PR)+2)), and the result is not stored. If the new contents of (((PR)+2)) are less than or equal to (((PR)+1)) a branch is executed to the location pointed to by ((PR)+3). Otherwise, the next instruction in sequence is executed.

Affected: ((((PR)+2)))

Condition Codes:

Z Set if (((PR)+2)) = ((PR)+1)

N Set if (((PR+2)) < ((PR)+1)

C Set if carry from most significant bit of result, otherwise cleared

O Set if operands were of opposite sign and the sign of the subtrahend was the same as the sign of the result, otherwise cleared

Enter Extended Memory Addressing Mode

The following instruction is used to enter the Classic Extended Memory Addressing Mode of CPU execution. This instruction may be programmed before any short-indexed or indexed instruction and will cause the operand address calculation to utilize a 21-bit extended address. The resultand EEA (Extended Effective Address) will be added to a modulo-8K lower boundary address stored in the EMCR (Extended Memory Control Register). The result will actually address the operand anywhere within the 2 megaword range of memory. The AEA must be less than or equal to the modulo-8K upper boundary address contained in the upper byte of the EMCR. For a detailed discussion of Extended Memory Addressing refer to the System Programming Addressing Mode section.

EXMA Enter Extended Memory Addressing

0 7	8 10 11	12 15
01	0 11	EMA

$(IR_{11\text{-}15})$ → EMA Register

(The next instruction performs operand addressing in the Extended Addressing mode.)

The next sequential instruction will use extended memory addressing for operand fetch/storage. This will apply to the next instruction only, reverting to normal mode thereafter.

Affected: EMA operand addressing mode for the next sequential instruction only

Condition Codes: No effect

Refer to page 8-16 for instructions that may use EXMA.

LXR

Enter/Exit Virtual Mode of CPU Execution

The following instructions are used to change the Central Processor Mode of execution from the initial Non-Virtual Modc to thc Virtual Mode, and from the Virtual Mode to the Non-Virtual Mode.

The normalized condition of the Classic (following POWER ON or MASTER CLEAR) disables the Memory Management System and Register Context System. Instruction execution will continue in the Non-Virtual Mode until the EVMO instruction is executed

The Classic then enters the Virtual Mode of execution. This operation invokes the virtual addressing logic with its associated page Access Rights protect scheme, and invokes the general register context logic which duplicates all modifications to the general purpose registers in the active program's Context Register Block.

Execution of the XVMO instruction returns Central Processor operation to the Non-Virtual Mode. The Memory Management System and Register Context System are disabled and execution continues in the Non-Virtual Mode until an EVMO is executed. The XVMO instruction is intended for use in diagnostics only.

Both the EVMO and XVMO instructions are privileged.

EVMO Enter Virtual Mode of CPU Execution (Privileged)

0 7	8 11	12 15
01	1	0

(Operation becomes virtual)

The CPU mode of operation becomes Virtual. The paged memory management and general register context logic is enabled.

Affected: CPU operating state

Condition Codes: Unaffected

XVMO Exit Virtual Mode of CPU Execution (Privileged)

0 7	8 11	12 15
01	8	0

(Operation becomes non-virtual)

The CPU mode of operation becomes non-virtual. The paged memory management and general register context logic is disabled.

Affected: CPU operating state

Condition Codes: Unaffected

Enter/Exit Pipelined Mode (Of Instruction Fetching)

The following two instructions are used to change the mode in which the Classic Central Processor reads instructions from memory. The Central Processor contains an instruction pipelining system which pre-calls instruction words for several consecutive locations beyond the current value of the Program Register (PR). The pre-called memory words are then stored in a First-In, First-Out (FIFO) instruction stack within the processor until ready for execution.

The look-ahead scheme is utilized differently in the Pipelined Mode and Non-Pipelined Mode of fetching instructions. In the Non-Pipelined Mode the instruction stack is flushed following every memory write operation, and its contents recalled. This insures that a modified memory location already precalled by the pipeline system will be recalled to utilize the new contents of that location.

In the Pipelined Mode the instruction stack will never be flushed because of a memory write. This substantially increases the effeciency of the pipeline system and imposes the software restraint that no write operation should address memory between (PR)+1 and (PR)+8 (the maximum possible look-ahead range of the pipeline).

The normalized condition of the Classic (following POWER ON or MASTER CLEAR) is the Non-Pipelined Mode. To enter the Pipelined Mode, the EPMD instruction is executed, which then disables instruction stack flushes due to memory write cycles. Stack flushes due to program branches and error conditions are still enabled. To return to the Non-Pipelined Mode, the XPMD instruction is executed.

Both the EPMD and XPMD instructions are Privileged.

EPMD Enter Pipelined Mode of Execution (Privileged)

0 7	8 11	12 15
E8	0	8

(Instruction Fetching is performed in Pipelined Mode)

The Central Processor is placed in the Pipelined Mode of fetching instructions and the next instruction in sequence is executed. The instruction stack is flushed when this instruction is executed.

Affected: None

Condition Codes: Not affected

XPMD Exit Pipelined Mode of Execution (Privileged)

0 7	8 11	12 15
E8	0	B

(Instruction Fetching returns to the Non-Pipelined Mode)

The Central Processor is taken out of the Pipelined Mode of Fetching instructions, the instruction stack is flushed and the next instruction is recalled before execution.

Affected None.

Condition Codes: Not affected

(Select and) Move Register Block

The following instructions are used to transfer individual context Register Blocks to and from memory ,and to select a context Register Block in the active Program Status Doubleword (PSD), then restore it to the general purpose registers.

The MRBM instruction is used to save the register environment of an interrupted task during context switching when the task's Register Block must be reassigned to a higher priority task. It also is used during power failure to save the volatile Register Context File - one block at a time.

The MMRB instruction is used to initially load the Register Context File during program startup and during power restoration. It also is used to provide a low overhead mechanism for restoring the register environment of an interrupted task when its context Register Block has been reassigned during the interruption.

The SCRB instruction is used to select the appropriate working Register Block in the active PSW and restore its contents to the general purpose registers as part of the context switch function to a higher priority task that has just interrupted program execution and requires CPU control.

The register-block-to-memory (MRBM) and memory-to-register-block (MMRB) operations use a general purpose register to specify the affected register block. The contents of that register can be obtained from the Program Status (word 1) of the active PSD of the appropriate task or interrupt using the affected Register Block.

Execution of these instructions is permissable only in the Privileged State. The Condition Codes are unaffected.

MRBM,r,x — Move Register-Block Section of Context File to Memory File (Privileged)

0 7	8 11	12 15
07	R	Rx

0 3	4 7	8 15	
x	RB	x	R

(RB1,RB2,... RB15) ⟶((Rx),(Rx)+1 ... (Rx)+14)

The contents of context Register Block RB replace the contents of short-indexed memory locations VEA thru VEA VEA+14.

Affected: (VEA, VEA+1, ... VEA+14)

Condition Codes: Unaffected

MMRB,r,x — Move Memory File to Register-Block Section of Context File (Privileged)

0 7	8 11	12 15
06	R	Rx

0 3	4 7	8 15	
x	RB	x	R

((Rx),(Rx)+1,... (Rx)+14) ⟶(RB1,RB2,... RB15)

The contents of short-indexed memory locations VEA thru VEA+14 replace the contents of context Register Block RB. The contents of the general purpose register file (GRF) are unaffected by this instruction. If an update of the current GRF is required, execution of this instruction must be followed by execution of an SCRB instruction.

Affected: (RB1, RB2, ... RB15)

Condition Codes: Unaffected

SCRB,r Select Current Register-Block in PSD (Privileged)

0 7	8 11	12 15
01	5	R

0 3	4 7	8 15	
x	RB	x	R

$$(R)_{4-7} \rightarrow (PS)_{4-7}$$

$$(RB1, RB2, \ldots RB15) \rightarrow (R1, R2, \ldots R15)$$

The contents of bits 4-7 of register R replace the contents of bits 4-7 in the Program Status (word 1) in the active PSD. The contents of context Register Block RB replace the contents of the General Purpose Register File.

Affected: $(PS)_{4-7}$, (R1, R2, . . . R15)

Condition Codes: Unaffected

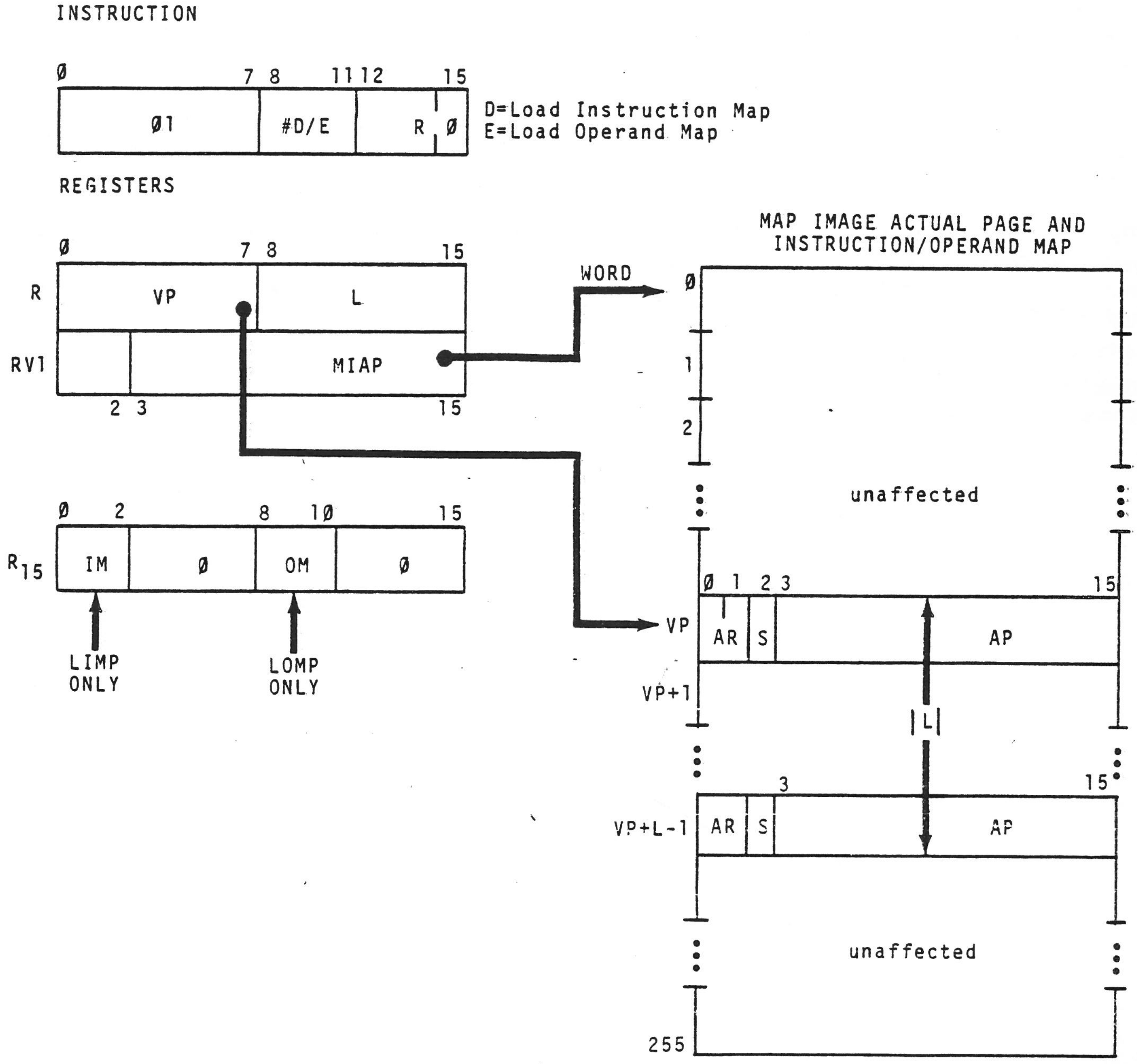
Load (Section Of) Map Image into Hardware Map

The following instructions are used to load a memory Map Image into a hardware Map. This must be done before a program can execute in the Virtual Mode. LIMP loads a hardware Instruction Map (IM) and LOMP loads a hardware Operand Map (OM).

LIMP and LOMP use a special form of addressing in which bits 12-14 of the instruction word point to a whole or partial Map and Map Image Address in registers R:RV1 and dedicated register R15.

Registers R:RV1 contain a 13-bit Map Image Actual Page number (MIAP) of the Map Image to transfer, a Virtual Page (VP) pointer to a relative word within both MIAP and IM or OM to begin the transfer, and a Length (L) designation of the number of MIAP words to transfer. The L designator is stored in an implied nine-bit two's complement format, where the most significant bit is always implied equal to one, allowing transfer counts in the range 1-256.

The hardware Map to be loaded is determined from either the IM or OM field of the program's Program Status word. This Map designator must be passed to LIMP or LOMP as an

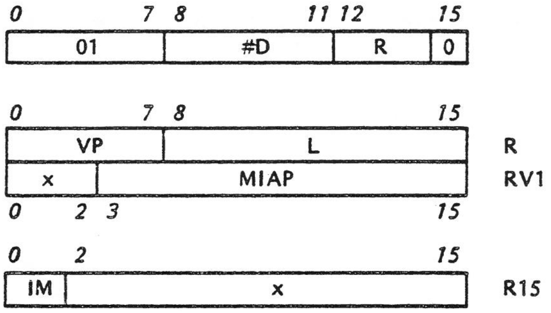


argument in dedicated general register R15. The LIMP instruction gets its argument from bits 0-2 of R15; the LOMP instruction uses bits 8-10. If a program only uses one memory Map, IM and OM are identical and either instruction can be executed. Both instructions must be executed for a program that uses two Maps.

Interrupt windows are opened following every 16 words transferred.

Execution of these instructions is permissible only in the Privileged State. The Condition Codes are unaffected.

LIMP,r — Load Instruction Map Image Into Hardware Map (Privileged)



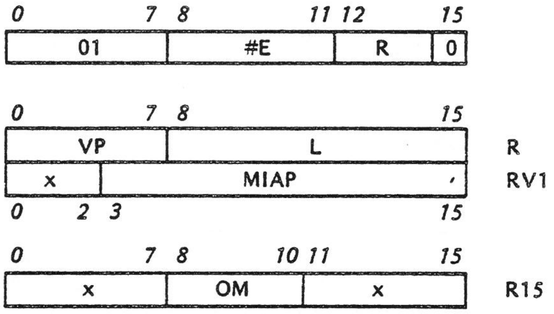
$\text{MIAP}_{\text{words VP-(VP+L-1)}} \longrightarrow \text{IM}_{\text{words VP-(VP+L-1)}}$

The contents of section VP thru VP+L-1 of Map Image MIAP, specified by registers R:RV1, replace the equivalent section of Instruction Map IM designated by bits 0-2 of register R15.

Affected: $(\text{IM})_{\text{VP-(VP+L-1)}}$, (R)

Condition Codes: Unaffected

LOMP,r — Load Operand Map Image Into Hardware Map (Privileged)



$\text{MIAP}_{\text{words V-(VP+L-1)}} \longrightarrow \text{OM}_{\text{words VP-(VP+L-1)}}$

The contents of section VP through VP+L-1 of Map Image MIAP, specified by registers R:RV1, replace the equivalent section of Operand Map OM designated by bits 8-10 of register R15.

Affected: $(\text{OM})_{\text{VP-(VP+L-1)}}$, (R)

Condition Codes: Unaffected

Store (Section Of) Hardware Map into Map Image

The following instruction is used to store the contents of a hardware map into a Map Image. This instruction is intended for diagnostic use and hence is noninterruptable.

SOMP uses a special form of addressing in which bits 12-14 of the instruction word point to a whole or partial Map and Map Image Address in registers R:RV1, and dedicated register 15.

Registers R and RV1 contain a 13 bit Map Image Actual Page number (MIAP) of the Map Image to be transferred into, Virtual Page (VP) pointer to a relative word within both MIAP and OM to begin the transfer and a length (L) designation of the number of Map words to transfer. The L designator is stored in an implied nine-bit two's complement format, where the most significant bit is always implied equal to one, allowing transfer counts in the range 1-256.

SOMP,r — Store Operand Map Into Map Image

0 7	8 11	12 14	15
01	#F	R	0

0 7	8 15	
VP	L	R

0 2	3 15	
x	MIAP	RV1

0 7	8 10	11 15	
x	OM	x	R15

$OM_{words\ VP\text{-}(VP+L-1)} \longrightarrow MIAP_{words\ VP\text{-}(VP+L-1)}$

The contents of section VP through VP+L-1 of operand map OM designated by bits 8-10 of register R15 replace the corresponding section of the Map Image specified by register RV1.

Affected: $MIAP_{VP\text{-}(VP+L-1)}$

Condition Codes:

N Reset
Z Set
O,C Unaffected

Zero Section of Hardware Map

The following instructions are used to clear those part(s) of a hardware Map that are not assigned to a program's virtual addressing space.

A program will possess memory Access Rights of 00 (no access or non-existent memory) for all zeroed (unassigned) Virtual Pages in its hardware Map. This allows a privileged program or operating system to use the equivalent parts of a program's Map Image in memory that are zeroes in its hardware Map, for other purposes (e.g., DMP I/O transfer information).

ZIMP clears a hardware Instruction Map (IM) and ZOMP clears a hardware Operand Map (OM).

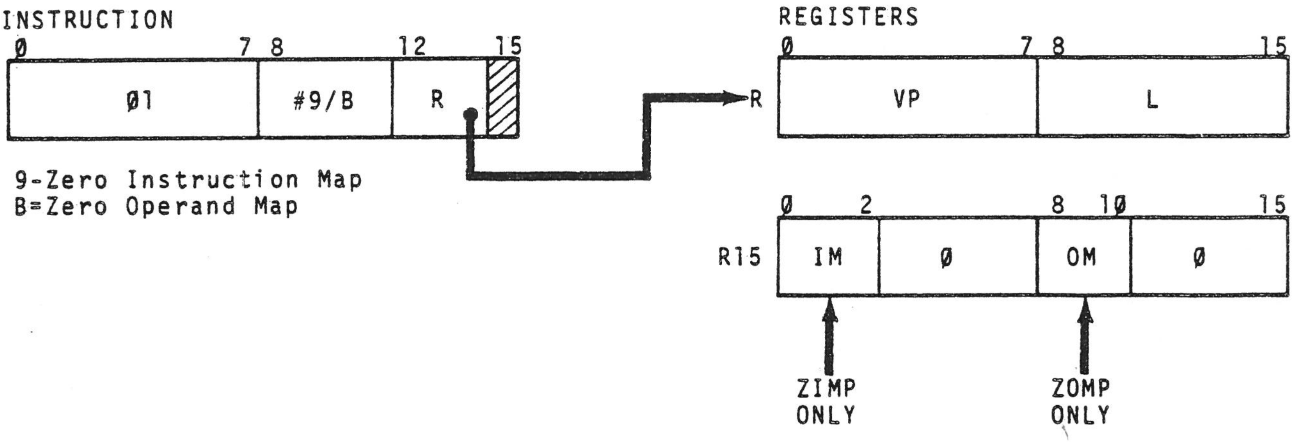
ZIMP and ZOMP use a special form of addressing in which bits 12-14 in the instruction word point to a whole or partial Map Address in register R and dedicated register R15.

Register R contains a Virtual Page (VP) pointer to a relative word within the hardware Map and a Length (L) designation of the number of Virtual Pages (Map registers) to clear.

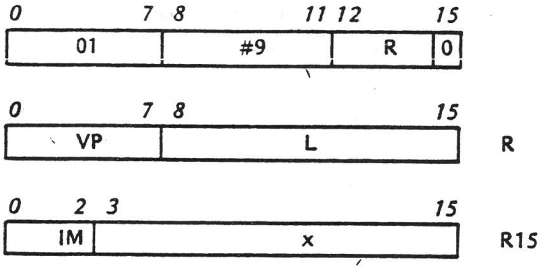
The hardware Map to be cleared is determined from the Program Status word of the program's PSD. This value must be available in register R15. The ZIMP instruction gets the desired map number from bits 0-2 of register R15 and ZOMP gets the desired map number from bits 8-10. If a program uses only one memory Map these values are identical.

See LIMP and LOMP Map and Map Image Addressing for additional discussion of this similar addressing mode. Interrupt windows are opened following every 16 words cleared.

Execution of these instructions is permissible only in the Privileged State. The Condition codes are unaffected.



ZIMP,r — Zero Section of Instruction Map (Privileged)



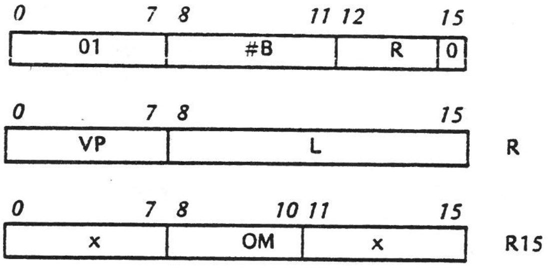
$0\text{'s} \rightarrow IM_{\text{words } VP\text{-}(VP+L-1)}$

Zero's replace the section, specified by register R, of Instruction Map IM designated by bits 0-2 of register R15.

Affected: $(IM)_{VP\text{-}(VP+L-1)}, (R)$

Condition Codes: Unaffected

ZOMP,r — Zero Section of Operand Map (Privileged)



$0\text{'s} \rightarrow OM_{\text{words } VP\text{-}(VP+L-1)}$

Zero's replace the section, specified by register R, of Operand Map OM designated by bits 8-10 of register R15.

Affected: $(OM)_{VP\text{-}(VP+L-1)}, (R)$

Condition Codes: Unaffected

Select Map as Current OM in PSD

The following instructions are used to select another program's Instruction Map (IM) or Operand Map (OM) as the current Operand Map in order to access its operands, then to reselect the operating system Map (Map 0) as the current Operand Map.

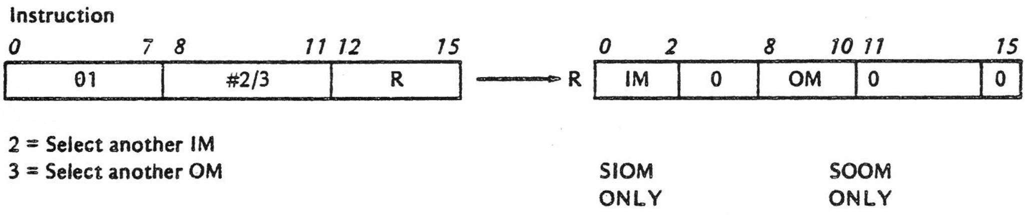
SIOM and SOOM permit a privileged program to temporarily select another program's virtual addressing space (hardware Map) as its own operand space. Using these instructions, it is then possible to fetch operands from the instruction space or operand space of the other program.

In order to access its addressing space the other program must be currently active (have a hardware Map assigned to it), which is the case when privileged service subroutines in MAP 0 which are called by the other program to run at its own task level (e.g., REX service subroutines) and perform privileged executive functions.

The program that uses SIOM and SOOM can no longer access its own operands (except for immediate instruction operands) until it reselects its original Operand Map.

SZOM permits such a program to restore its own operand Map 0 when a privileged service (or other) function is complete. SIOM or SOOM can be used to accomplish this function if a program's own Operand Map is not Map 0.

SIOM and SOOM use a special form of addressing, in which bits 12-15 of the instruction word point to a Map Selector in register R. The Instruction Map (IM) or Operand Map (OM) that is to become the current Operand Map is defined in the Program Status word of the program whose map(s) are to be selected. SIOM selects the desired map using bits 0-2 of Register R; SOOM selects the desired map using bits 8-10.



2 = Select another IM
3 = Select another OM

Execution of these instructions is permissible only in the Privileged State. The Condition Codes are not affected.

SIOM,r — Select Another Program's IM As Current OM (Privileged)

0 7	8 11	12 15
01	#2	R

0 2	3 15	
IM	x	R

$(R)_{0-2} \longrightarrow (PS)_{8-10}$

Instruction Map IM designated by bits 0-2 of register R is selected as the current Operand Map in the Active Program Status Doubleword.

Affected: $(PS)_{8-10}$

Condition Codes: Unaffected

SOOM,r — Select Another Program's OM As Current OM (Privileged)

0 7	8 11	12 15
01	#3	R

0 7	8 10	11 15
x	OM	x

$(R)_{8-10} \longrightarrow (PS)_{8-10}$

Operand Map OM designated by bits 8-10 of register R is selected as the current Operand Map in the Active Program Status Doubleword.

Affected: $(PS)_{8-10}$

Condition Codes: Unaffected

SZOM — Select Map Zero As Current OM (Privileged)

0 7	8 11	12 15
01	#4	0

0's $\longrightarrow (PS)_{8-10}$

Map 0 is selected as the current Operand Map in the active Program Status Doubleword.

Affected: $(PS)_{8-10}$

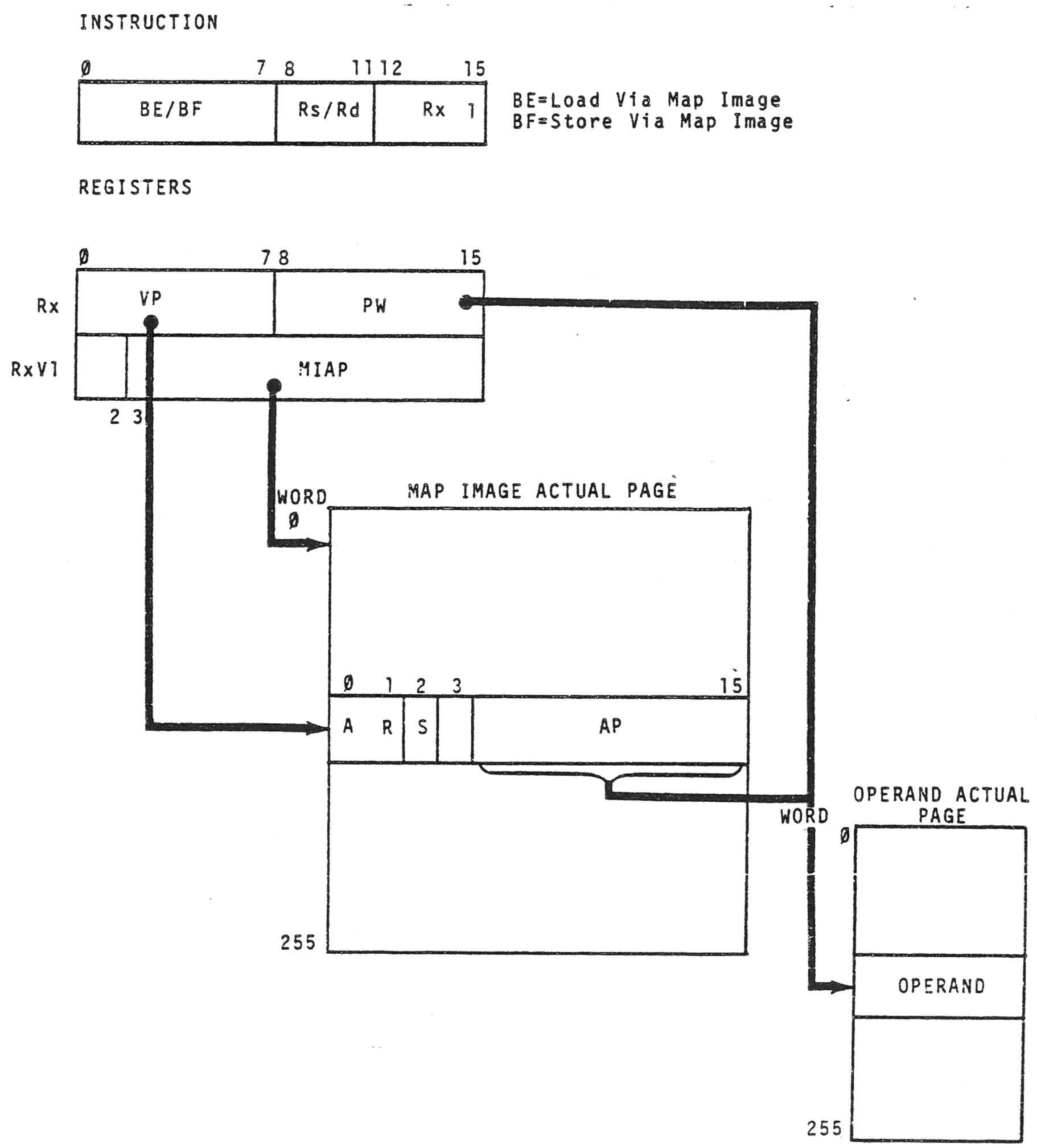
Condition Codes: Unaffected

Load/Store Word (Via Map Image)

The following instructions are used to invoke memory addressing via a Map Image, thus bypassing the virtual addressing imposed by hardware memory Maps. These memory maps are used when transporting operands between the General Purpose Registers and memory in the Virtual Mode of execution.

Memory addressing via a Map Image is useful for transferring operands to/from the virtual addressing space of a program that is suspended and does not have its map image loaded in any hardware Map. Interrupt driven I/O handlers are examples of routines which might need to access such operands asynchronous with task execution. These instructions operate identically in Virtual or Non-Virtual machine mode.

LDVM (load) and STVM (store) use a special form of operand addressing in which bits 12-14 of the instruction word point to an operand Via a Map Image Address in registers Rs:RsV1. These registers contain a 13-bit Map Image Actual Page number (MIAP), a Virtual Page (VP) pointer to a relative word within the MIAP containing the Actual Page number, and a Page word (PW) number within the Actual Page to load or store. The specified Map Image need not be mapped in any virtual space.



BE=Load Via Map Image
BF=Store Via Map Image

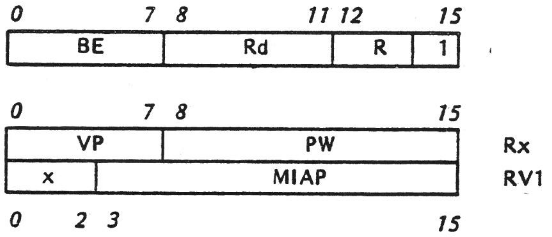
Register Rx contains a 16-bit Virtual Address to be translated into an 21-bit Actual Address (AA) by table look-up via a Map Image in Actual Memory. Register RxV1 contains the Map Image Actual Page number.

AA forms the final Effective memory Actual Address of the operand.

The 16-bit contents of location AA will be loaded either into register Rd (LDVM) or will have register Rd stored into it (STVM). If the page Access Rights (AR) indicate nonexistent memory (AR=00), operand access will not occur and the overflow condition is set. Operand access also will not occur if AR ≠ 11, causing the carry condition to be set.

Execution of these instructions is permissible only in the Privileged State.

LDVM,r,r — Load Register From Memory (Via Map Image) (Privileged)



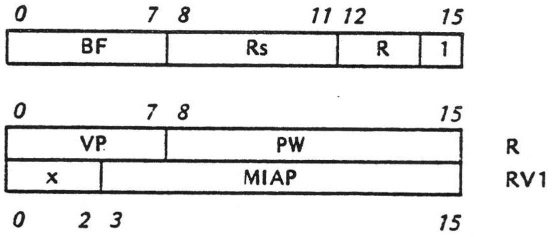
$((RV1)_{3-15}:(R)_{0-7})_{3-15}:(R)_{8-15}) \longrightarrow (Rd)$

The contents of Actual Memory Location AA addressed via a Map Image replace the contents of register Rd.

Affected: (Rd)

Condition Codes:
- Z Set if (Rd) = 0
- N Set if (Rd) <0
- C Set if Access Rights do not = 11, otherwise cleared
- O Set if Access Rights = 00, otherwise cleared

STVM,s,r — Store Register Into Memory (Via Map Image) (Privileged)



$(Rs) \longrightarrow ((RV1)_{3-15}:(R)_{0-7})_{3-15}: (R)_{8-15}$

The contents of register Rs replace the contents of Actual Memory location AA addressed via a map image. (Note: This instruction must not alter any memory locations between (PR)+1 and (PR)+8 within its own addressing space.

Affected: (AA)

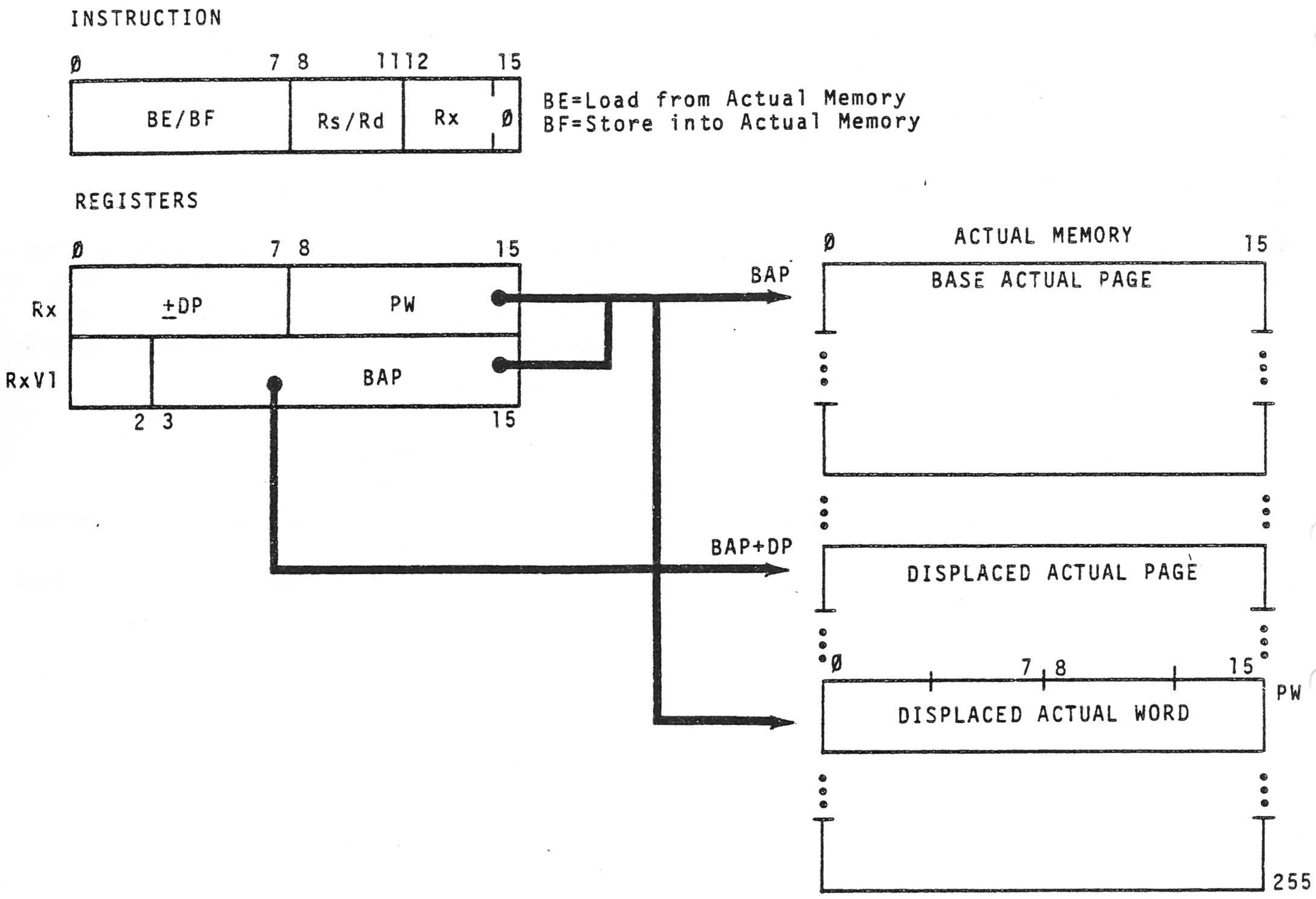
Condition Codes:
- Z Unaffected
- N Unaffected
- C Set if Access Rights do not = 11, otherwise cleared
- O Set if Access Rights = 00, otherwise cleared

Load/Store Word in (Actual) Memory

The following instructions are used to invoke displaced addressing of Actual Memory, thus bypassing the virtual addressing imposed by hardware memory Maps when transferring operands between the general purpose registers and memory in the Virtual Mode of execution.

Actual Memory addressing is useful to access memory operands directly in Actual Memory, and operates identically in Virtual or Non-Virtual machine mode. The instructions are particularly useful for addressing Actual Memory above 64K words in MODCOMP II Mode and for accessing large shared memory modules used as data banks in either execution mode.

LDAM (load) and STAM (store) use a special form of operand addressing in which bits 12-14 of the instruction word point to an Actual Memory Address in registers Rx:RxV1. Register RxV1 contains a 13-bit pointer to a Base Actual Page (BAP) in Actual Memory. Register Rx contains a signed Displacement Page (DP) and a Page Word (PW) which point to a Displaced Actual Word (DAW) within ±32,768 Actual Memory locations of word 0 in the Base Actual Page.

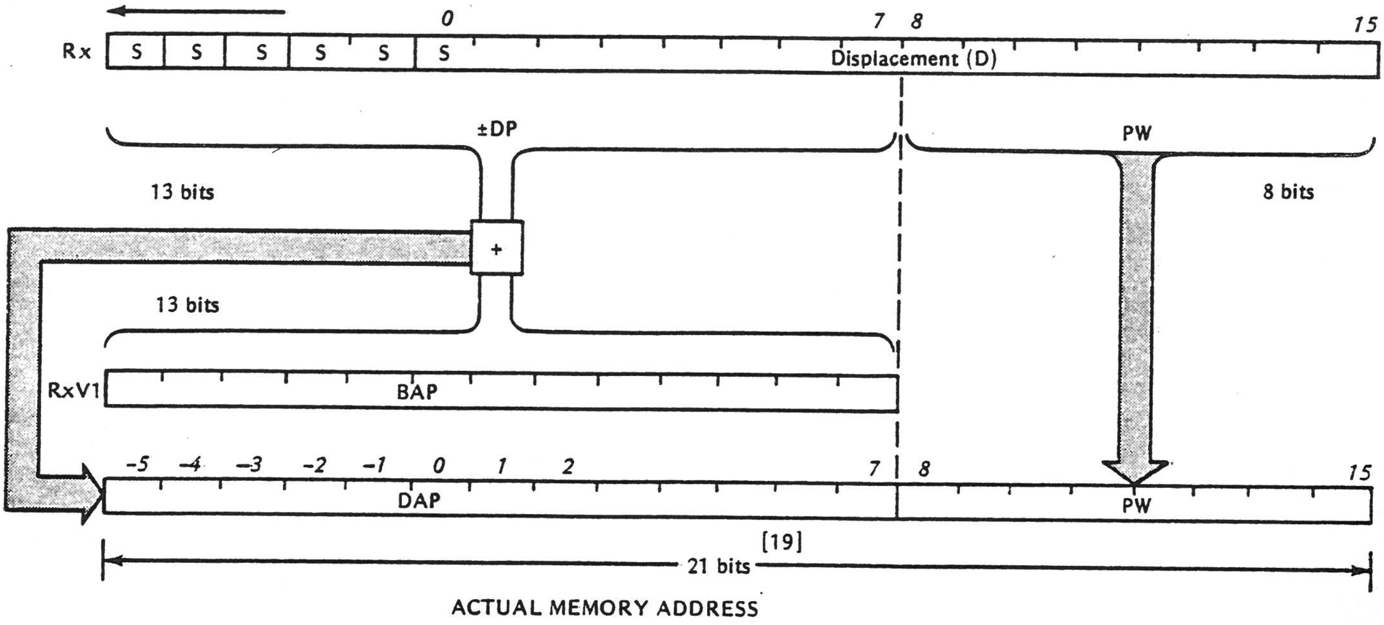


BE=Load from Actual Memory
BF=Store into Actual Memory

The Displacement Page and Page Word pointers in Register Rx form a 16-bit signed Displacement value (D), with five assumed additional high-order extended sign bits. The high-order 13-bits (DP) of the extended Displacement Value are added to the BAP to form the Displaced Actual Page (DAP) number. The resulting 13-bit DAP is contatenated with the 8-bit PW to form the 21-bit Actual Memory Address (AA).

The signed Displacement in Register Rx functions not only as an index pointer but also can be a limit count when negative.

Execution of these instructions is permissible only in the Privileged State. The LDAM instruction affects the Condition Codes.



LDAM,r,x — Load Register From (Actual) Memory (Privileged)

0	7	8	11	12	15
BE		Rd		Rx	0

0	15	
←Displacement (D)		Rx
x	AP	RxV1
0 2	3 15	

$((RxV1)_{3-15}+(Rx)_{0-7}:(Rx)_{8-15}) \longrightarrow (Rd)$

The contents of Actual Memory location AA replace the contents of register Rd.

Affected: (Rd)

Condition Codes:

Z	Set if (Rd) = 0
N	Set if (Rd) < 0
C	Unaffected
O	Unaffected

STAM,s,x — Store Register Into (Actual) Memory (Privileged)

0	7	8	11	12	15
BF		Rs		Rx	0

0	15	
←Displacement (D)		Rx
x	AP	RxV1
0 2	3 15	

$(Rs) \longrightarrow ((RxV1)_{3-15}+(Rx)_{0-7}:(Rx)_{8-15})$

The contents of register Rs replace the contents of actual memory location AA. This instruction must not alter any memory location between (PR)+1 and (PR)+8 within its own addressing space.

Branch to Microroutine

These instructions allow the programmer to enter an arbitrary point in microcode. They are intended for use in diagnostics but may also be used to implement custom macro instructions. They are privileged instructions.

BRM,r Branch to Microroutine (Privileged)

0 7	8 11	12 15
0E	R	E

0 3	4 15	
0	Micro Location	R

$(R_{4\text{-}15}) \rightarrow$ MPC (Micro Program Counter)

The low order 12 bits of the contents of register R are loaded into the CPU micro program counter. Execution then resumes at that point.

Affected: MPC

Condition Codes: Unaffected

BRMI ML Branch to Microroutine Immediate (Privileged)

0 7	8 11	12 15
0E	0	F
0	Micro Location	

$((P)+1)_{4\text{-}15} \rightarrow$ MPC (Micro Program Counter)

The low order 12 bits of the immediate memory location are loaded into the CPU micro program counter. Execution then resumes at that point.

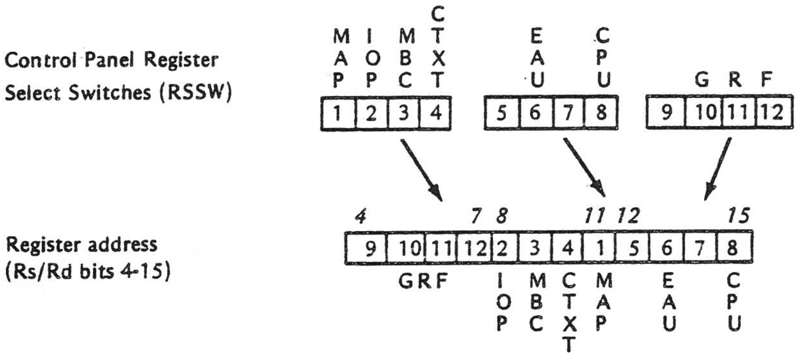
Affected: MPC

Condition Codes: Unaffected

Read/Write Internal Registers

These instructions provide the capability to read or write any hardware register available from the control panel. These instructions are privileged, and are intended for use in diagnostics. The GRF is not an allowable source/destination register.

The register address, in Rs/Rd, is in the following format:


Control Panel Register Select Switches (RSSW)
MAP IOP MBC CTXT
1 2 3 4
EAU CPU
5 6 7 8
GRF
9 10 11 12
Register address (Rs/Rd bits 4-15)
4 7 8 11 12 15
9 10 11 12 2 3 4 1 5 6 7 8
GRF IOP MBC CTXT MAP EAU CPU

RDI,r A — Read Internal Register (Privileged)

0 7	8 11	12	15
OE	Rd	0	C

0 3	4 15	
0	Register Address	Rd
Register Contents		RdV1

$(R_{(Rd)}) \rightarrow (RdV1)$

The contents of the register addressed by contents of register Rd are loaded into register RdV1. A register in the GRF may not be sourced.

Affected: (RdV1)

Condition Codes: Not affected

WRI,r A — Write Internal Register (Privileged)

0 7	8 11	12	15
OE	Rs	0	D

0 3	4 15	
	Register Address	Rs
Register Contents		RsV1

$(RsV1) \rightarrow (R_{(Rs)})$

The contents of register RsV1 are transferred to the register addressed by the contents of register Rs. A register in the GRF may not be destined.

Affected: $(R_{(Rs)})$

Condition Codes: Not Affected

Read/Write Memory Status

The following instructions are used to write or read the contents of various memory status registers present on each physical memory module in the system. These registers track data and address during normal transfers and latch upon detection of certain memory error conditions.

Condition codes are not affected by these instructions.

RMPS,r,x Read Memory Plane Status (Privileged)

0 7	8 11	12	15
03	Rd	Rx	0

(Memory Plane Status Register) selected by $(RxV1)_{11-13}$ and $(Rx)_{11-15}$: $(RxV1)_{14-15}$ → (Rd)

The contents of register Rd are replaced by the contents of a memory plane status register whose address is contained in Rx:RxV1. The contents of Rx:RxV1 have the following format:

0 10	11 15	
Not Used	EMA	Rx

0 10	11 13	14 15	
Not Used	Reg ##	Low Order MA Bits	RxV1

Affected: (Rd)

Condition Codes: Not Affected

RMWS,r,x Read Memory Word Status (Privileged)

0 7	8 11	12	15
03	Rd	Rx	1

(Memory Word Status Register), selected by $(Rx)_{11-15}$: (RxV1) → (Rd)

The contents of register Rd are replaced by the contents of the memory word status register whose address is contained in Rx:RxV1. Rx:RxV1 have the following format:

0 10	11 15	
Not Used	EMA	Rx

0 15	
Memory Address	RxV1

Affected: (Rd)

Condition Codes: Not Affected

Note: Refer to Chapter 3 for additional information.

WMS,s,x Write Memory Status (Privileged)

0 7	8 11	12	15
04	Rs	Rx	0

The contents of register Rs replace the contents of a memory status register whose address is contained in registers Rx:RxV1. The WMS instruction is used to either reset the error bit contained in a memory plane status register or write parity/check bit information into a memory word status register and corresponding actual memory word addressed by Rx:RxV1. Error detection and correction mode control bits may also be written via the WMS command. Register 0 should not be selected as a source register for the WMS instruction. It will cause the instruction to no-op if selected.

Rx:RxV1 have the following format:

0 10	11 15	
Not Used	EMA	Rx

0 15	
Memory Address	RxV1

Affected: (Memory word status register) and/or (Memory plane status register)

Condition Codes: Not affected

Note: Refer to Chapter 3 for additional information.

Interrupt and Call

This instruction class provides the capability for interrupt status interrogation and for complete program manipulation of the states of the three Request, Enable and Active latches present in each of 16 hardware priority interrupt levels.

Additional capability is provided to clear the highest interrupt level Active (and, optionally, Request) latch(es) and return execution to the previous highest level.

Capability also is provided to invoke calls on operating system service routines in Map 0, and to enable each CPU in a multiprocessor configuration to produce an interrupt in the other CPU.

All instructions in this class, except those that request an executive service and interrogate interrupt status, are privileged and may be executed only in the Privileged State. The Condition codes are not affected.

Request Executive Service

The following instruction is used to request an operating system executive service. To accomplish this, an interrupt request signal is sent to the Unimplemented Instruction Trap level. The executive service requested is defined by the contents of the Service field. The program count is not advanced before the trap is generated. Therefore, the stored PSD contains the address of the REX service.

The Unimplemented Instruction Trap level will become active and the interrupt routine entered at the completion of execution of the REX instruction, provided that neither this level nor any higher level is already active. If this level or a higher level is active, execution of the REX cannot be completed. The machine must be manually cleared and restarted if this error condition occurs.

REX,s Request Executive Service

0 7	8 15
23	Service

$1 \longrightarrow REQ_{UI}$

The Request latch of the Unimplemented Instruction priority interrupt level (UI) is set. If this level is highest, the requested Service will be performed by the corresponding function of the interrupt routine connected to that level.

Affected: $(REQ)_{UI}$

Condition Codes: Unaffected

Load Register with Current Interrupt Latches

The following instructions are used to load the status of the interrupt Request, Enable and Active latches into a general purpose register.

LCIE,r — Load Register with Current Interrupt Enable Latches

0 7	8 11	12 15
0E	Rd	5

$(ENA)_G \rightarrow (Rd)$

The Enabled latches of the priority interrupt levels are transferred to register Rd.

Affected: (Rd)

Condition Codes: Unaffected

LCIR,r — Load Register with Current Interrupt Request Latches

0 7	8 11	12 15
0E	Rd	6

$(REQ)_G \rightarrow (Rd)$

The Request latches of the priority interrupt levels are transferred to register Rd.

Affected: (Rd)

Condition Codes: Unaffected

LCIA,r — Load Register with Current Interrupt Active Latches

0 7	8 11	12 15
0E	Rd	4

$(ACT)_G \rightarrow (Rd)$

The Active latches of the priority interrupt levels are transferred to register Rd.

Affected: (Rd)

Condition Codes: Unaffected

Set Interrupt Latches

The following instructions are used to set the interrupt Enable, Request and Active latches at a selected interrupt level.

SIE,I Set Interrupt Enable (Privileged)

0 7	8 11	12 15
26	4	Level

$1 \rightarrow (\mathrm{ENA})_{G,\ \mathrm{Level}}$

The priority interrupt level specified by the Level selection field is enabled. (Note: The Power FailSafe/AutoStart, Memory Parity, Unimplemented Instruction, System Protect and Floating Point Overflow Trap interrupt levels are always enabled when present in the system. The Enable latch state of these levels cannot be altered by instruction execution.)

Affected: None

Condition Codes: Unaffected

SIR,I Set Interrupt Request (Privileged)

0 7	8 11	12 15
26	8	Level

$1 \rightarrow (\mathrm{REQ})_{G,\ \mathrm{Level}}$

The Request latch of the priority interrupt level specified by the Level selection is set. (Note: Levels C_{16} and D_{16} should not be requested by the program.)

Affected: None

Condition Codes: Unaffected

Execution of these instructions is privileged and may occur only in the Privileged State.

SIA,I Set Interrupt Active (Privileged)

0 7	8 11	12 15
26	0	Level

$1 \rightarrow (\mathrm{ACT})_{G,\ \mathrm{Level}}$

The priority interrupt level specified by the Level selection field is activated. (Note: Level 0 (Power FailSafe/AutoStart) may not be set active by the program.)

Affected: None

Condition Codes: Unaffected

Reset Interrupt Latches

The following instructions are used to reset the Enable, Request, and Active latches at a selected interrupt level.

RIE,I Reset Interrupt Enable (Privileged)

0 7	8 11	12 15
27	4	Level

$0 \rightarrow (ENA)_{G,Level}$

The priority interrupt level specified by the Level selection field is disabled. (Note: The Power FailSafe/AutoStart, Unimplemented Instruction, System Protect and Floating Point Overflow Trap interrupt levels are always enabled when present in the system. The Enable latch state of these levels cannot be altered by instruction execution.)

Affected: None

Condition Codes: Unaffected

RIR,I Reset Interrupt Request (Privileged)

0 7	8 11	12 15
27	8	Level

$0 \rightarrow (REQ)_{G,Level}$

The Request latch of the priority interrupt level specified by the Level selection field is reset.

Affected: None

Condition Codes: Unaffected

Execution of these instructions is privileged and may occur only in the Privileged State.

RIA,I Reset Interrupt Active (Privileged)

0 7	8 11	12 15
27	0	Level

$0 \rightarrow (ACT)_{G,Level}$

The priority interrupt level specified by the Level selection field is deactivated.

Affected: None

Condition Codes: Unaffected

Clear Interrupt (Request) and Return to Previous Level

The following instructions are used to clear the Active latch (and optionally the Request latch) of the highest active interrupt level and return task execution either to the previous highest level before interruption, or to a new machine state.

Both CAR and CIR clear the Active latch of the highest active interrupt level. CIR also clears the Request latch of the highest active interrupt level. Then both instructions load the Program Status doubleword (PSD) from either the dedicated locations of the restored highest active interrupt level (old PR and old PS) or from registers Rs:RsV1. A return to the original machine state that existed before the interruption, or the invocation of a new machine state, is thus effected.

No interrupt can go into service until at least one machine instruction has been executed after the CAR instruction.

If the Request latch remains set continuously, CAR may be used to return from an interrupt trace routine which traces every instruction of the program it constantly interrupts. If no interrupt is active when the CAR instruction is executed, the dedicated locations 0 (PR) and 1 (PS) are used to restore the PSD if register Rs = 0.

An interrupt cleared by the CIR instruction, which resets the Request latch, will not go into service again until a subsequent setting of its Request latch occurs (if still enabled). If no interrupt is active when the CIR instruction is executed, the dedicated locations 0 and 1 are used to restore the PSD if register Rs = 0.

CAR,s — Clear Active and Return (Privileged)

0 7	8 11	12 14	15
24	0	Rs	0

```
                           0   →(ACT)Highest
If (Rs) = 0,:   (OLD PR)       →(PR)
                (OLD PS)       →(PS)
If (Rs) does not = 0: (Rs)     →(PR)
                     (RsV1)    →(PS)
```

The Active latch of the highest active interrupt level is cleared. The Program Register and Program Status contained in the dedicated memory locations of the previous high active interrupt level, or contained in registers Rs:RsV1, replace the contents of the current Program Status Doubleword.

Affected: $(ACT)_{Highest}$, (PSD)

Condition Codes: Unaffected

CIR,s — Clear Interrupt and Return (Privileged)

0 7	8 11	12 14	15
25	0	Rs	0

```
                             0  →(ACT)Highest
                             0  →(REQ)Highest
If (Rs)  = 0:   (OLD PR)        →(PR)
                (OLD PS)        →(PS)
If (Rs) does not =0:  (Rs)      →(PR)
                     (RsV1)     →(PS)
```

The active and Request latches of the highest active interrupt level are cleared. The Program Register and Program Status contained in the dedicated memory locations of the previous highest active interrupt level, or contained in registers Rs:RsV1, replace the contents of the current Program Status Doubleword.

Affected: $(ACT)_{HIGHEST}$, $(REQ)_{HIGHEST}$, (PSD)

Condition Codes: Unaffected

Request Multiprocessor Interrupt(s)

The following instruction is used to request a Multiprocessor Interrupt on any one or more of four separate lines. Each line may be connected to the Multiprocessor Interrupt level of another CPU.

From one to four separate pulses are generated by the executing CPU, each of which request the interprocessor communications interrupt in the connected CPU(s). The associated response to this pulse in the interrupted CPU(s) is a function of the interrupt service routine connected to its Multiprocessor Interrupt level.

RMI,I Request Multiprocessor Interrupt(s) (Privileged)

0 7	*8 11*	*12*			*15*
01	0	L1	L2	L3	L4

$1 \rightarrow (\text{REQ})_{\text{MI}}$ IN CPU $_{\text{Line(s)}}$

The Request latch of the priority Multiprocessor Interrupt level (3) is set in CPU L1, L2, L3 and/or L4.

Affected: $(\text{REQ})_{\text{MI}}$ in CPU $_{\text{Line(s)}}$

Condition Codes: Unaffected'

Input/Output Instructions

This instruction class provides the capability to enable a data or status word to be transferred from any peripheral device to any General Register, and to enable a data or command word to be transferred from any General Register to any peripheral device.

Since some peripheral device controllers transfer data only under control of the Direct Memory Processor (DMP), only command and status words are transferred under program control to/from these devices when I/O is performed in the DMP Mode. A DMP initialization instruction is provided for all DMP I/O transfers performed in the MODCOMP IV Virtual Mode.

All of the instructions in this class are privileged.

Up to 64 peripheral device controllers, consisting of four groups of 16 each, are addressable by the register data, command and status instructions in the Register I/O mode. Up to 63 uniquely addressable peripheral device controllers may be configured when utilizing DMP capabilities, as DMP channel #00 is used by all DMP controllers during the DMP sequence. The group address is obtained from the two least significant bits of the operation code field.

I/O GROUP A	Consists of device addresses 00-0F
I/O GROUP B	Consists of device addresses 10-1F
I/O GROUP C	Consists of device addresses 20-2F
I/O GROUP D	Consists of device addresses 30-3F

Two signals control I/O command decode and are interrogated at I/O sync time (DRIOSN). They are:

DRIOFN – Input/Output Function
DRCDFN – Command/Data Function

Data flow is determined by DRIOFN. If DRIOFN is true (low), data is input. If DRIOFN is false (high), data is output.

Instruction type is determined by DFCDFN. If DFCDFN is true (low), the instruction is data. If DRCDFN is false (high), the instruction is a command (or status).

	DRCDFN	$\overline{\text{DRCDFN}}$
DRIOFN	Input Data IDA	Input Status ISA
$\overline{\text{DRIOFN}}$	Output Data ODA	Output Command OSA

Further considerations in the programming of these I/O functions are described at the end of this section. A description of the I/O instructions is presented first.

Input Status From Device Zero

The following instruction is used by system software to determine its operating environment. It transfers a status word from the I/O Processor to the Central Processor, which contains the following information.

MC IV/25	0
/35	8000
Classic	2000

0	1	2	3 … 15
T	M	C	0

Bit	*Status*	

0	TYPE –	Has meaning only if bit 2 = 0
		T=0: IV/25 processor
		T=1: IV/35 processor

1	MODE–	Has meaning only if processor is IV/35 or V (T=1 and C=0, or C=1)
		M=0: non-relocatable mode
		M=1: relocatable mode

2	CLASS–	C=0: MODCOMP IV processor
		C=1: CLASSIC processor

ISZ,r Input Status From Device Zero

0 … 7	8 … 11	12 … 15
48	Rd	0

(IOP Status) → (Rd)

The I/O Processor status word replaces the contents of Rd.

Affected: (Rd)

Condition Codes: Unaffected

ISA,B,C,D

Input Status From I/O Group A,B,C, or D

The following instruction is used to input the status of a peripheral device in I/O Group A, B, C, or D into a general purpose register.

Execution of this instruction causes the contents of the 16 data lines to be transferred to the specified register. The controller, as selected by the device address, puts its status word on the data lines and then the transfer occurs.

One basic format exists which is common to all controllers. This format encompasses two groups: Errors and Events. The error group has a pointer bit indicating if any error is set. The status format is so defined that a status word of all zeroes is invalid, indicating a malfunctioning or non-existent controller.

0	1	2	3	4	5	6	7	8	9	15
E	D	P	I	M			B	T		

Error Field (bits 0–6) Event Field (bits 7–15)

Bit *Status*

0 Error Pointer Bit
Zero - An error has occured and is defined in the field of bits 1 through 6
One - No error has occured

1 Data transfer error
Zero - No error
One - Overflow or underflow error

2 Parity or checksum error
Zero - No error
One - Device parity error

3 Inoperable
Zero - Device operable (on-line)
One - Device inoperable (off-line, interlock open, etc.)

4 Memory violation error
Zero - No error
One - A memory violation or parity error was detected during a DMP transfer

5-6 Specify error conditions unique to a device such as seek error

7 Busy status of device controller
Zero - Device controller not busy
One - Device controller busy

8 Transfer Status. (Normally used when not operating in the interrupt mode)
Zero - Device controller ready to transfer a data word
One - Device controller not ready to transfer a data word

9-15 Specify device unique event conditions

ISA,B,C,D,r D Input Status From I/O Group A,B,C, or D (Privileged)

0 3	4 5	6 7	8 11	12 15
4	10_2	G	Rd	D

G,D → (I/O Address Lines)
Device Status → (Rd)

Operation	*Code (bits 0-7)*
ISA	48
ISB	49
ISC	4A
ISD	4B

The group (G) and device (D) numbers contained in the instruction word are placed on the I/O bus address lines.

Up to 16 bits of status are then transferred from the addressed device over the I/O bus to replace the contents of register Rd.

Affected: (Rd), (I/O Address Lines)

Condition Codes:
Z Set if all status bits are zero, otherwise cleared
N Set if status bit 0 is set, otherwise cleared
C Set if status bit 8 is set, otherwise cleared
O Set if status bit 7 is set, otherwise cleared

Input Data From I/O Group A,B,C or D

The following instruction is used to input data from a peripheral device in I/O Group A, B, C, or D into a general purpose register. Data input under control of this instruction does not invoke the Direct Memory Processor.

IDA,B,C,D,r D Input Data from I/O Group A,B,C, or D (Privileged)

0 3	*4 5*	*6 7*	*8 11*	*12 15*
4	11_2	G	Rd	D

G,D → (I/O Address Lines)
Device Data → (Rd)

Operation	*Code (bits 0-7)*
IDA	4C
IDB	4D
IDC	4E
IDD	4F

The group (G) and device (D) numbers contained in the instruction word are placed on the I/O bus address lines.

Up to 16 bits of data are then transferred from the addressed device over the I/O bus to replace the contents of register Rd.

Affected: (Rd), (I/O Address Lines)

Condition Codes: NZOC

1000 if byte in bits 8-15 is (
1001 if byte is)
1010 if byte is *
1011 if byte is +
1100 if byte is ,
1101 if byte is -
1110 if byte is .
1111 if byte is /
0000 if byte is alphabetic
0001 No Special Code
0011 if byte is space
0101 if byte is numeric

Output Command to I/O Group A,B,C or D

The following instruction is used to output a command to a peripheral device in I/O Group A, B, C, or D from a general purpose register.

Execution of this instruction Transfers the 16-bit output command stored in the specified register to the I/O register where it is placed on the I/O bus data lines.

There are three basic command formats: Select, Control and Transfer Initiate. The bit designations for each group are defined below. All standard peripheral controllers follow these format conventions. All Commands except End-of-Block and Terminate reset all stored status if the device is not busy. The standard controllers interpret the command as follows:

Select Command Format:

0	*1*	*2*	*15*
0	0		

Bits *Function*

0-1 Must both be zero. These bits specify the select format

2-15 Specify a set-up condition such as unit number (multi-unit controllers), density, head number, etc.

Control Command Format:

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*15*
0	1	D	S	E	T	M P E		

Bit *Function*

0 Must be zero.

1 Must be one. This bit is used in conjunction with bit 0 to specify the control format.

2 Specifies the state of the data interrupt:
Zero - Disconnects the device controller and resets the request in the controller if present.
One - Connects the device controller sub-level to the data interrupt level.
If the DMP mode had previously been specified by a Transfer Initiate command, the data interrupt will occur when the Word Count is exhausted.
If the register I/O mode had previously been specified by a Transfer Initiate command, the data interrupt is interpreted as Data Request.

3 Specifies the state of the service interrupt.
Zero - Disconnects and resets the request if active.
One - Connect the interrupt, allowing it to become active.
The service interrupt may be caused by a variety of conditions such as end of record or error. The interrupt condition depends on controller design.

4 Specifies End-of-Block command when equal to one. No effect when equal to zero. The End-of-Block command causes the controller (except the Teletype Controller) to immediately generate a data interrupt if that interrupt had previously been connected. This function is useful for diagnostic and debugging purposes. An End-of-Block command will be accepted even when a controller is busy or operating in the DMP mode.

The responding device ignores all bits of the control format except 0, 1, 4 and 5.

5 Specifies Terminate command when equal to one. No effect when equal to zero. The Terminate command stops data transfer to/from the specified device and resets any non-active data interrupt, or DMP Data request. A Terminate command will be accepted when a controller is busy or in the DMP mode.

The Terminate command will also condition a controller to generate a service interrupt when the controller is subsequently ready to respond to another Transfer Initiate command. If the controller is not busy and the service interrupt has been previously connected, this interrupt will occur immediately. The responding device will ignore all bits of the control format except 0, 1, 4 and 5.

If a terminate command is issued with bit 7 set to one, a controller with DMP facilities will set its memory violation error status indicator. This command is normally issued automatically by the DMP I/O system if such an error is detected. It indicates either a memory addressing violation or a memory parity error.

If bit 7 is set to one within a terminate command to to some devices (TTY for example), an immediate operation abort occurs.

If bit 9 is set within a terminate command, some devices are caused to Master Clear.

Bit	*Function*
6	Normally used to distinguish between a normal control command and a No-Op. See No-Op below.
7	Specify a control function such as rewind, advance record, seek cylinder, etc. . . otherwise, as already noted.

No-Op Command

When bits 4, 5, and 6 are all zero, bits 7-15 are ignored. The No-Op command alters interrupt connection per the values of bits 2 and 3 whether or not the device is busy. The No-Op command also resets all device status if the device is not busy.

Interrupt Disconnection and Termination

An interrupt may be reset by means of a Disconnect or Terminate command whenever the interrupt level is Active. However, if the level is not active but might become active immediately, an invalid request might occur on the I/O level. To accommodate this situation, requests at levels 80 and CO should execute a CIR and return to the interrupted program.

Transfer Initiate Command Format:

0	1	2	3	4	5 — 15
1	M	D	S	I	

Bits	*Function*
0	Must be one. This bit specifies the Transfer Initiate Format.
1	Specifies mode selection for subsequent data transfer. Zero - Sets the device to the programmed register I/O mode. The device sends a data interrupt request, if connected, each time it requires a data transfer, including the first transfer. One - Sets the device to the DMP transfer mode.
2	Specifies the state of the data interrupt. Zero - Disconnects the device controller and resets the request in the controller if present. One - Connects the device controller sub-level to the data interrupt level. If the DMP mode had previously been specified by a Transfer Initiate command, the data interrupt will occur when the TC = 0. If the register I/O mode had previously been specified by a Transfer Initiate command, the data interrupt is defined as Data Request.
3	Specifies the state of the service interrupt. Zero - Disconnects and resets the request if active. One - Connects the interrupt, allowing it to become active. The service interrupt may be caused by a variety of conditions such as the end of the record or error. The interrupt condition depends on controller design.
4	Specifies the direction of data transfer. Zero - Sets the device to the output transfer mode. One - Sets the device to the input transfer mode.
5-15	Specifies a transfer initiate function such as write record or read card.

OCA,B,C,D,s D — Output Command to I/O Group A, B, C or D (Privileged)

0 — 3	4 — 5	6 — 7	8 — 11	12 — 15
4	00_2	G	Rs	D

G,D → (I/O Address Lines)
(Rs) → (I/O Data Lines)

Operation	*Code (Bits 0-7)*
OCA	40
OCB	41
OCC	42
OCD	43

The group (G) and device (D) numbers contained in the instruction word are placed on the I/O bus address lines.

The 16 bit output command stored in register Rs is then transferred to the I/O register and placed on the I/O bus data lines.

Affected: (I/O Address and Data Lines)

Condition Codes: Unaffected

Output Data to I/O Group A, B, C or D

The following instruction is used to output data to a peripheral device in I/O Group A, B, C, or D from a general purpose register. Data output under control of the instruction does not invoke the Direct Memory Processor.

ODA,B,C,D,s D Output Data to I/O Group A,B,C, or D (Privileged)

0 3	*4 5*	*6 7*	*8 11*	*12 15*
4	01_2	G	Rs	D

G,D → (I/O Address Lines)
(Rs) → (I/O Data Lines)

Operation	*Code (Bits 0-7)*
ODA	44
ODB	45
ODC	46
ODD	47

The group (G) and device (D) numbers contained in the instruction word are placed on the I/O bus address lines.

The 16-bit data word stored in register Rs is then transferred to the I/O register and placed on the I/O bus data lines.

Affected: (I/O Address and Data Lines)

Condition Codes: Unaffected

Initialize Direct Memory Processor

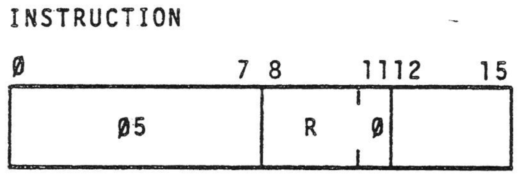
The following instruction is used to inform a DMP channel of the memory Map Image it will use to define the virtual addressing space for its subsequent I/O data transfer operations.

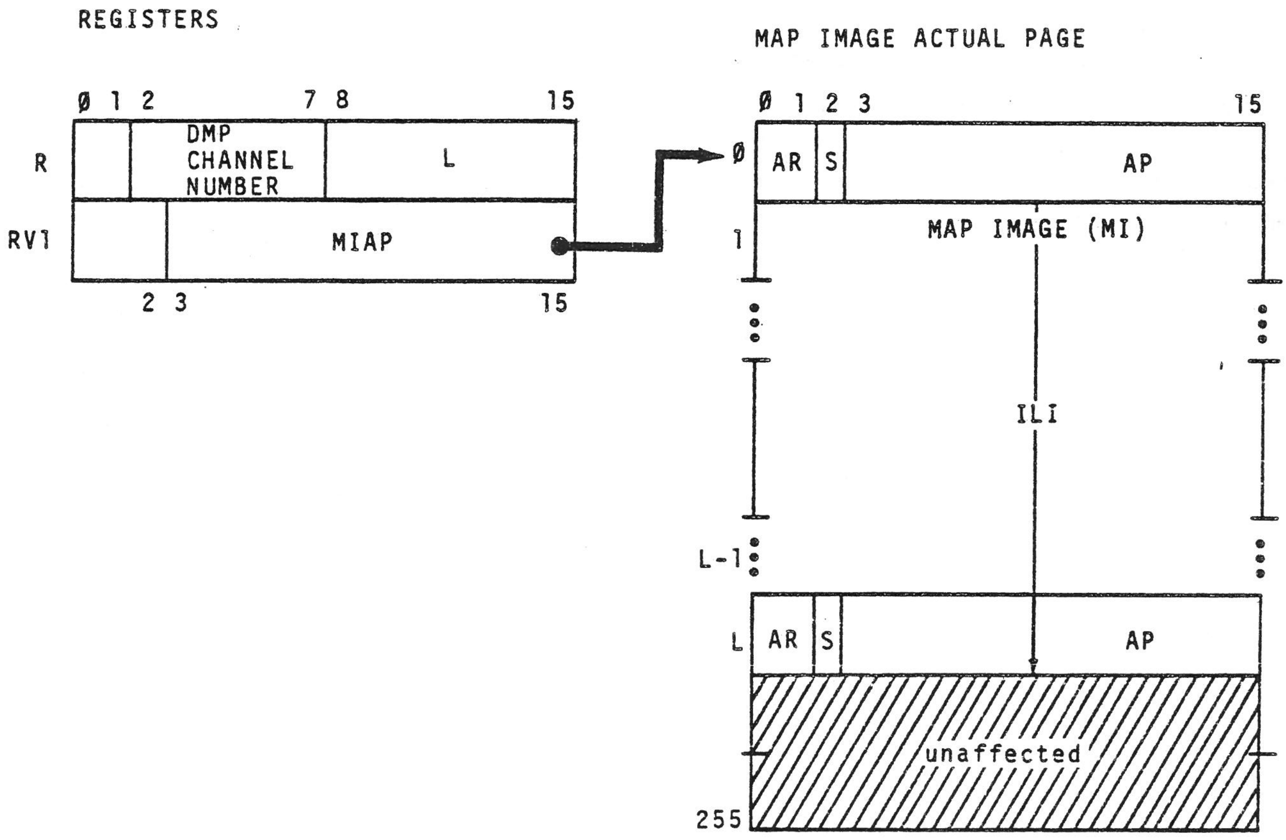
All Transfer Addresses (TA) which define the start of data buffers, and all Chain Addresses (CA) which define the start of scattered buffer sections, are virtual. These Virtual Addresses are translated by the DMP channel hardware to Actual Addresses dynamically using the specified Map Image, while the I/O operation is being performed.

The DMP channel hardware also interprets and polices the Access Rights of all pages in which data buffer sections are defined, in order to protect privileged memory areas.

The Map Image Actual Page in which all data buffer sections and data string chain addresses are defined must remain available in Actual Memory and its DMP page allocations unchanged for the duration of the I/O operation. This allows the program for which the DMP I/O operation is performed to become suspended while its I/O takes place, and will assure the program of its virtual addressing space.

DMPI uses a special form of addressing in which bits 8-10 of the instruction word point to a whole or partial DMP Map Image Address in registers R:RV1. Register R contains the DMP channel number right-justified in its most significant byte. The least significant byte contains the 8-bit L parameter which designates the number of Virtual pages available. Register R+1 contains the 13-bit MIAP number in bits 3-15 and the associated access rights bits in 0,1.





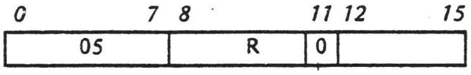
Register RV1 contains the 13-bit Actual Page address of the CMP channel Map Image. Register R contains an 8-bit Length (L) value, in two's complement form, which limits the size of the Map Image to be all or part of an entire page. Thus, when a program size is such that it does not need an entire page of the Map Image to define all of its virtual address space, the program or the operating system can use the remainder of the page for some other purpose. See AMEM and DMEM Map Image Addressing for additional discussion of this similar addressing mode.

Since the DMP will honor the Access Rights (AR) of each translated Virtual Address, any violation of an Access Right will abnormally terminate the I/O operation in progress. Any Virtual Addresses which attempt to use the Map Image above the Virtual Address specified by the length L will be treated as if they were assigned an Access Right of 00 (no access, non-existent memory). The Memory Parity Error (MPE) status bit in the device status (bit 5) will indicate this form of abnormal termination.

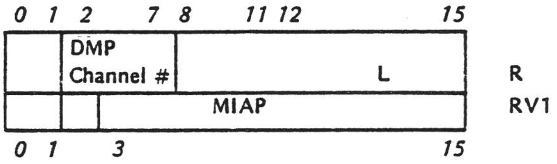
The DMPI instruction must be executed before the Transfer Initiate command is given for the I/O Unit/Group using the DMP when the Classic is executing in the Virtual Mode. This instruction is ignored in the non-virtual MODCOMP II Mode.

Virtual Address translation overhead in the DMP is limited to a single extra memory access of the Map Image, only when a data buffer crosses a memory page boundary, or when the first word of a new data buffer section or data chain is accessed. Hardware registers in the DMP channel retain the Actual Page addresses being used for all other memory transfers. These extra accesses, as all accesses made by the DMP, are invisible if the buffer areas are not in the same memory module with the currently executing program.

DMPI,r C Initialize Direct Memory Processor (Privileged)



$(R{:}RV1)_{MIAP,L} \rightarrow (DMP)_{Channel\ \#\ Registers}$



The Map Image Actual Page address (MIAP) and its useable Length (L) contained in register R:RV1 are transmitted to the specified DMP channel (DMP#). This information initializes the DMP channel with the starting location and limit of the control memory area containing the buffer sections and data string chains to be used in the subsequent I/O data transfer operations.

Affected: DMP Channel

Condition Codes: Unaffected

I/O Programming Considerations

The sequence of programming steps necessary to perform typical register and DMP input/output functions are described here. The descriptions are general purpose and are designed to cover all contingencies.

Register I/O Interrupt Mode Sequence

New Command Initiation

Store interrupt subroutine starting address in the data and service interrupt level dedicated locations.

Reset previous error and interrupt status by the execution of an Output Command instruction with a No-Op output command, disconnecting both interrupts.

Test present device status by executing an Input Status instruction.

If status indicates inoperable or an invalid (all zero) status word, exit to error routine; otherwise continue.

Execute an Output Command Instruction specifying transfer initiate, register mode, input or output, connection of interrupts and any other modifiers required by the particular peripheral device. Exit and wait for interrupt.

The controller is now busy and will not respond to new initiation commands. It will respond to Input Status, Input Data and Output Data Instructions and an Output Command Instruction with a Terminate Command. The controller will produce the data interrupt when a data transfer is required and the service interrupt if a malfunction occurs or at the end of the media record.

Response to Data Interrupt

The data interrupt processing routine is automatically entered when the requesting controller has data ready and the highest priority.

Preserve original contents of R1 - execute an STM, R1, A. Repeat for all other registers to be used as working registers.

Check Word Count

If transfer not complete, perform input or output operation as required. If output, load new data into appropriate place in register, execute Output Data Instruction and update word and byte counts appropriately. If input, execute execute Input Data Instruction and move or store data as required before updating word and byte counts.

If the last word required was transferred, an Output Command Instruction should be executed, issuing a Terminate Command to the controller. This will stop further transfers and reset the data interrupt request.

Restore the previous contents of the working registers.

Execute a CIR Instruction to exit the data interrupt routine and return to the original program.

Response to Service Interrupt

The Service Interrupt Processing routine is automatically entered if the requesting controller has the highest priority. This interrupt is generated after all hardware checks are complete and the controller can accept a new initiation command.

- Preserve the original contents of R1 - execute an STM, R1, A. Repeat for all registers to be used as working registers.
- Check validity of the transfer by issuing an Input Status instruction. If an abnormality is indicated, exit to error recovery routine.
- If previous checks are satisfactory and no further tasks are required for controller, restore the previous contents of the working registers and execute a CIR.
- If previous checks are satisfactory and another transfer sequence is desired, execute an Output Command Instruction with a new initiation command. This command will reset any status conditions that may be set.

Register I/O Test and Transfer Mode

Register I/O transfers may be accomplished without the use of data interrupts. A "Data Ready" bit is provided in the standard status word for this purpose. To operate in this mode, the data interrupt is disconnected by the initiation command and the data ready bit is tested during each transfer sequence. Device control is performed in the same manner as in the interrupt mode.

Direct Memory Processor I/O Mode

The optional DMP mode frees the program from the task of handling individual data word transfers, and increases net throughput capabilities. The software initiation and termination sequences are described in this section in the most general manner possible.

New Command Initiation

Store interrupt subroutine starting addresses in the two dedicated interrupt level locations.

Reset previous error or interrupt status by execution of an Output Command, which also disconnects both interrupt levels.

Test present status by executing an Input Status Instruction. If inoperability is indicated or an invalid (all zero) status word, exit to error routine.

Store a transfer address and a word count in the two DMP dedicated locations.

Execute a DMPI instruction to transfer the starting location of the address mapping list and its length.

Execute an Output Command Instruction with an Initiate Command specifying DMP mode, input or output, connection of service interrupt, optional connection of data interrupt, and any other modifiers required by the particular peripheral.

Exit and wait for the interrupt.

The controller is now busy and will not respond to new initiation commands. It will respond only to an Input Status Instruction or an Output Command Instruction with a Terminate or No-Op Command.

Response to Data Interrupt

The data interrupt processing routine is automatically entered when the controller requesting has highest priority.

Preserve the original contents of R1 - execute an STM, R1, A. Repeat as required for all working registers.

The occurrence of this interrupt, in this mode, designates that the transfer of the block of data has been completed (TC=0). The program may use this fact to gain time to manipulate data prior to the completion of the physical media operation.

Execute a CIR, which will return control to the point of interruption.

Response to Service Interrupt

The service interrupt processing routine is automatically entered when the requesting controller is the highest in the interrupt queue.

This interrupt is generated after all hardware status checks are complete and the controller is ready to accept a new initiation command.

Preserve the original contents of R1 - execute an STM, R1, A. Repeat for all other registers to be used as working registers.

Check validity of the transfer by executing an Input Status instruction. If an abnormality is indicated, exit to an error recovery routine.

Check final transfer address in dedicated location. If improper, exit to error routine.

If the previous checks are satisfactory and no further tasks are required, restore working registers and execute a CIR instruction to return to the original program.

If the previous checks are satisfactory and another block transfer is desired, load the word count and transfer address into the dedicated locations. Then execute an Output Command Instruction with a new Initiation Command. This command will reset any status conditions that may be set.

Execute a CIR instruction to exit the routine.

Appendix A. DEDICATED MEMORY LOCATION MAP

0	1F	Not Used	32
20	3F	1st Group PI's (16)	32
	5F	MCII Mode and Status Storage Ded. Loc's for MCIV	32
60	7F	DMP TC and TA Ded. Loc's	32
	FF	IO Party Line PI Vectors	128
100	1FF	Map Zero - Map Image	256
200		Not Dedicated	

Appendix B. CHARACTER CODES

ASCII		Card Code		EBCDIC		Compressed Alpha-Numeric		TTY		Line Printer	
ASCII CHAR.	HEX. Code	029 Key Punch Symbol	ANSI Code	Char.	HEX. Code	COMP. Char.	CAN Code	TTY	HEX. Code	L.P. Symbol	HEX. Code
NUL	00	M.P.	12-0-9-8-1	NUL	00			NUL	80		
SOH	01	M.P.	12-9-1	SOH	01			SOM	81		
STX	02	M.P.	12-9-2	STX	02			EOA	82		
ETX	03	M.P.	12-9-3	ETX	03			EOM	83		
EOT	04	M.P.	9-7	EOT	37			EDT	84		
ENQ	05	M.P.	0-9-8-5	ENQ	2D			WRU	85		
ACK	06	M.P.	0-9-8-6	ACK	2E			RU	86		
BEL	07	M.P.	0-9-8-7	BEL	2F			BEL	87		
BS	08	M.P.	11-9-6	BS	16			FE$_0$	88		
HT	09	M.P.	12-9-5	HT	05			HT	89		
LF	0A	M.P.	0-9-5	LF	25			LF	8A		
VT	0B	M.P.	12-9-8-3	VT	0B			VT	8B		
FF	0C	M.P.	12-9-8-4	FF	0C			FORM	8C		
CR	0D	M.P.	12-9-8-5	CR	0D			RETURN	8D		
S0	0E	M.P.	12-9-8-6	S0	0E			S0	8E		
SI	0F	M.P.	12-9-8-7	SI	0F			SI	8F		
DLE	10	M.P.	12-11-9-8-1	DLE	10			DC0	90		
DC1	11	M.P.	11-9-1	DC1	11			X-ON	91		
DC2	12	M.P.	11-9-2	DC2	12			TAPE	92		
DC3	13	M.P.	11-9-3	DC3	13			X-OFF	93		
DC4	14	M.P.	9-8-4	DC4	3C			$\overline{\text{TAPE}}$	94		
NAK	15	M.P.	9-8-5	NAK	3D			ERROR	95		
SYN	16	M.P.	9-2	SYN	32			SYC	96		
ETB	17	M.P.	0-9-6	ETB	26			LEM	97		
CAN	18	M.P.	11-9-8	CAN	18			S0	98		
EM	19	M.P.	11-9-8-1	EM	19			S1	99		
SUB	1A	M.P.	9-8-7	SUB	3F			S2	9A		
ESC	1B	M.P.	0-9-7	ESC	27			S3	9B		
FS	1C	M.P.	11-9-8-4	S4	9C			FS	1C		
GS	1D	M.P.	11-9-8-5	GS	1D			S5	9D		
RS	1E	M.P.	11-9-8-6	RS	1E			S6	9E		
US	1F	M.P.	11-9-8-7	US	1F			S7	9F		
SPACE	20	SPACE BAR	(NO PUNCH)	SPACE	40	SPACE	0	SPACE	A0	SPACE	20
!	21	|	12-8-7	!	4F			!	A1	!	21
"	22	"	8-7	"	7F			"	A2	"	22
#	23	#	8-3	#	7B			#	A3	#	23

M.P. = Multi-Punch (No-Symbol)

ASCII		Card Code		EBCDIC		Compressed Alphanumeric		TTY		Line Printer	
ASCII Char.	HEX. Code	029 Key Punch Symbol	ANSI Code	Char.	HEX. Code	COMP. Char.	CAN Code	TTY	HEX. Code	L.P. Symbol	HEX. Code
$	24	$	11-8-3	$	5B	$	39	$	A4	$	24
%	25	%	0-8-4	%	6C			%	A5	%	25
&	26	&	12	&	50			&	A6	&	26
'	27	'	8-5	'	7D			'	A7	'	27
(	28	(	12-8-5	(	4D			(	A8	(	28
)	29	)	11-8-5	)	5D			)	A9	)	29
*	2A	*	11-8-4	*	5C			*	AA	*	2A
+	2B	+	12-8-6	+	4E			+	AB	+	2B
,	2C	,	0-8-3	,	6B			,	AC	,	2C
—	2D	—	11	—	60			—	AD	—	2D
.	2E	.	12-8-3	.	4B		38	.	AE	.	2E
/	2F	/	0-1	/	61			/	AF	/	2F
0	30	0	0	0	F0	0	27	0	B0	0	30
1	31	1	1	1	F1	1	28	1	B1	1	31
2	32	2	2	2	F2	2	29	2	B2	2	32
3	33	3	3	3	F3	3	30	3	B3	3	33
4	34	4	4	4	F4	4	31	4	B4	4	34
5	35	5	5	5	F5	5	32	5	B5	5	35
6	36	6	6	6	F6	6	33	6	B6	6	36
7	37	7	7	7	F7	7	34	7	B7	7	37
8	38	8	8	8	F8	8	35	8	B8	8	38
9	39	9	9	9	F9	9	36	9	B9	9	39
:	3A	"	8-2	"	7A	:	37	:	BA	:	3A
;	3B	;	11-8-6	;	5E			;	BB	;	3B
<	3C	<	12-8-4	<	4C			<	BC	<	3C
=	3D	=	8-6	=	7E			=	BD	=	3D
>	3E	>	0-8-6	>	6E			>	BE	>	3E
?	3F	?	0-8-7	?	6F			?	BF	?	3F
@	40	@	8-4	@	7C			@	C0	@	40
A	41	A	12-1	A	C1	A	1	A	C1	A	41
B	42	B	12-2	B	C2	B	2	B	C2	B	42
C	43	C	12-3	C	C3	C	3	C	C3	C	43
D	44	D	12-4	D	C4	D	4	D	C4	D	44
E	45	E	12-5	E	C5	E	5	E	C5	E	45
F	46	F	12-6	F	C6	F	6	F	C6	F	46
G	47	G	12-7	G	C7	G	7	G	C7	G	47
H	48	H	12-8	H	C8	H	8	H	C8	H	48
I	49	I	12-9	I	C9	I	9	I	C9	I	49
J	4A	J	11-1	J	D1	J	10	J	CA	J	4A
K	4B	K	11-2	K	D2	K	11	K	CB	K	4B
L	4C	L	11-3	L	D3	L	12	L	CC	L	4C
M	4D	M	11-4	M	D4	M	13	M	CD	M	4D
N	4E	N	11-5	N	D5	N	14	N	CE	N	4E

M.P. = Multi-Punch (No-Symbol)

ASCII		Card Code		EBCDIC		Compressed Alphanumeric		TTY		Line Printer	
ASCII Char.	HEX. Code	029 Key Punch Symbol	ANSI Code	Char.	HEX. Code	COMP. Char.	CAN Code	TTY	HEX. Code	L.P. Symbol	HEX. Code
O	4F	O	11-6	O	D6	O	15	O	CF	O	4F
P	50	P	11-7	P	D7	P	16	P	D0	P	50
Q	51	Q	11-8	Q	D8	Q	17	Q	D1	Q	51
R	52	R	11-9	R	D9	R	18	R	D2	R	52
S	53	S	0-2	S	E2	S	19	S	D3	S	53
T	54	T	0-3	T	E3	T	20	T	D4	T	54
U	55	U	0-4	U	E4	U	21	U	D5	U	55
V	56	V	0-5	V	E5	V	22	V	D6	V	56
W	57	W	0-6	W	E6	W	23	W	D7	W	57
X	58	X	0-7	X	E7	X	24	X	D8	X	58
Y	59	Y	0-8	Y	E8	Y	25	Y	D9	Y	59
Z	5A	Z	0-9	Z	E9	Z	26	Z	DA	Z	5A
[	5B	¢	12-8-2	¢	4A			[	DB	[	5B
\	5C	\	0-8-2	\	E0			\	DC	\	5C
]	5D	!	11-8-2	!	5A			]	DD	]	5D
Δ	5E	¬	11-8-7	∧	5F			↑	DE	∧	5E
—	5F	—	0-8-5	—	6D			←	DF	—	5F
	60	M.P.	8-1		79						
a	61	M.P.	12-0-1	a	81						
b	62	M.P.	12-0-2	b	82						
c	63	M.P.	12-0-3	c	83						
d	64	M.P.	12-0-4	d	84						
e	65	M.P.	12-0-5	e	85						
f	66	M.P.	12-0-6	f	86						
g	67	M.P.	12-0-7	g	87						
h	68	M.P.	12-0-8	h	88						
i	69	M.P.	12-0-9	i	89						
j	6A	M.P.	12-11-1	j	91						
k	6B	M.P.	12-11-2	k	92						
l	6C	M.P.	12-11-3	l	93						
m	6D	M.P.	12-11-4	m	94						
n	6E	M.P.	12-11-5	n	95						
o	6F	M.P.	12-11-6	o	96						
p	70	M.P.	12-11-7	p	97						
q	71	M.P.	12-11-8	q	98						
r	72	M.P.	12-11-9	r	99						
s	73	M.P.	11-0-2	s	A2						
t	74	M.P.	11-0-3	t	A3						
u	75	M.P.	11-0-4	u	A4						
v	76	M.P.	11-0-5	v	A5						
w	77	M.P.	11-0-6	w	A6						
x	78	M.P.	11-0-7	x	A7						

M.P. = Multi-Punch (No-Symbol)

ASCII		Card Code		EBCDIC		Compressed Alphanumeric		TTY		Line Printer	
ASCII Char.	HEX. Code	029 Key Punch Symbol	ANSI Code	Char.	HEX. Code.	COMP. Char.	CAN Code	TTY	HEX. Code	L.P. Symbol	HEX. Code
y	79	M.P.	11-0-8	y	A8						
z	7A	M.P.	11-0-9	z	A9						
{	7B	M.P.	12-0	{	C0						
¦	7C	M.P.	12-11	¦	6A						
}	7D	M.P.	11-0	}	D0						
~	7E	M.P.	11-0-1	~	A1						
DEL	7F	M.P.	12-9-7	DEL	07			RUB OUT	FF		

M.P. = Multi-Punch (No-Symbol)

Appendix C. PERIPHERAL DEVICE COMMANDS AND STATUS

MOVING HEAD DISC

First Device Address 01_{16} DMP 1

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TRANSFER INITIATE	WRITE	1	M	CONN. DI	CONN. SI	0	EOD	EOF	IEOR	SCS	IGNORED		SECTOR				
	READ	1	M	CONN. DI	CONN. SI	1	IGNORED			SCS	IGNORED		SECTOR				
CONTROL	TERM/ EOB	0	1	IGNORED		EOB	TERM	IGN	MPE	IGNORED							
	CYLINDER SELECT	0	1	CONN. DI	CONN. SI	0	0	1	CYLINDER								
	HEAD DRIVE	0	0	IGN				PLAT	HEAD	CONT. SCAN	IGNORED				1 = PREP MODE	UNIT	
	NO-OP	0	1	CONN. DI	CONN. SI	0	0	0	IGNORED								
STATUS		0 =* ERROR	1 = UF OF	1 =* CRC	1 = INOP	1 =* MPE	1 =* WLD	1 =* SEEK	1 = BUSY	0 = DATA READY	EOD	EOF	EOR	1 = DEV. SEEK	1 = SEEK COMP.	UNIT	

M = Mode, if Bit 1 = 0, programmed I/O selected rather than DMP

*NOTE: If Bit 0 = 1, Bits 2, 4, 5, and 6 have the following meanings:

2 = 200 TPI	5 = Write Protected
4 = 5727 Controller	6 = Dual Platter

FIXED HEAD DISC (CONTROLLER ADDRESS 02_{16})

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TRANSFER INITIATE	WRITE	1	M	CONN. DI	CONN. SI	0	EOD	EOF	IGNORED				S	S	S	S	S
	READ	1	M	CONN. DI	CONN. SI	1	IGNORED						S	S	S	S	S
CONTROL	HEAD SELECT	0	1	CONN. DI	CONN. SI	0	0	1	IGN	H	H	H	H	H	H	H	H
	TERM/ EOB	0	1	IGNORED		EOB	TERM.	IGN.	MPE	IGNORED							
	UNIT* SELECT	0	0	IGNORED												U	U
	NO-OP	0	1	CONN. DI	CONN. SI	0	0	Ø		IGNORED							
	STATUS	0= ERROR	O/F U/F	CK SUM	IN OP	MPE	WLO	Ø	1= BUSY	0= DATA READY	EOD	EOF	EOR	0	0	U	U

S = SECTOR, H = HEAD, DMP LOCATIONS: TC = 62, TA = 72

U = UNIT NO., UP TO 4 UNITS MAY BE CONNECTED TO A CONTROLLER

M - MODE, IF BIT 1 = 0 PROGRAMMED I/O IS SELECTED RATHER THAN DMP

* THIS COMMAND NOT USED WITH MODEL 4103, 4104, 4105.

MAGNETIC TAPE (DEVICE ADDRESS, HIGH SPEED 03_{16}, LOW SPEED 04_{16})

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TRANSFER INITIATE	WRITE	1	M	CONN. DI	CONN. SI	0	0= BINARY 1= INTER-CHANGE	0= N-GAP 1= L-GAP	1= SINGLE CYCLE SCAN	0= ODD 1= EVEN	0= 800 CPI 1= 556 CPI	0	0	0	0	U	U
	READ	1	M	CONN. DI	CONN. SI	1	0= BINARY 1= INTER-CHANGE	0	1=SCS	0= ODD 1= EVEN	0= 800 CPI 1= 556 CPI	0	0	0	0	U	U
CONTROL	WRITE EOF	0	1	CONN. DI	CONN. SI	0	0	1	1=SCS	0	0= 800 CPI 1= 556 CPI	WRITE EOF 1	0	0	0	U	U
	SPACE	0	1	CONN. DI	CONN. SI	0	0	1	1=SCS	0	0= 800 CPI 1= 556 CPI	0	SPACE 1	0= FORWARD 1= REVERSE	0= BLOCK 1= FILE	U	U
	EOB/ TERM	0	1	IGN	IGN	1=EOB	1=TERM	IGN	1=MPE	IGNORED							
	REWIND	0	1	CONN. DI	CONN. SI	0	0	1	1= LOCK OUT	REWIND 1	0	0	0	0	0	U	U
	TRANSPORT SELECT	0	1	CONN. DI	CONN. SI	0	0	1	1= CONT. SCAN	0	0	0	0	0	0	U	U
	NO-OP	0	1	CONN. DI	CONN. SI	0	0	0	IGNORED								
STATUS		0= ERROR	1= OVER/ FLOW OR B>C	1= DEVICE PARITY ERROR	1= IN OP	1= MEMORY PARITY ERROR	1= FILE PROTECT	1= TAPE DEFECT	1= CONT. BUSY	0= DATA READY	1= EOT	1= EOF	1= BOT	1= DEVICE OFFLINE OR REWIND	1= PARTIAL WORD	U	U

U = UNIT NO., UP TO 4 UNITS MAY BE CONNECTED TO A CONTROLLER
M = MODE, IF BIT 1 = 0 PROGRAMMED I/O IS SELECTED RATHER THAN DMP
DMP LOCATIONS (HEX.)   TC=63, TA=73   TC=64, TA=74

DMP CARD READER (DEVICE ADDRESS 06_{16} DMP 6)

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TRANSFER INITIATE	READ	1	M	CONN DI	CONN SI	1	C0	C1	1 = PACKED	0	1 = ATT. INT.	0	0	0	0	0	0
CONTROL	ATTENTION READY	0	1	CONN DI	CONN SI	0	0	1	0	0	0	0	0	0	0	0	0
	TERM/ EOB	0	1	0	0	EOB	TERM	0	MPE	0	MC	IGNORED					
	SECONDARY STATUS	0	0	0	0	0	0	0	0	1	IGNORED						
	PROGRAM WRITE	1	M	CONN DI	CONN SI	0	0	1	0	0	0	0	0	0	0	0	1
	NO-OP	0	1	CONN DI	CONN SI	IGNORED											
STATUS		0 = ERROR	1 = DMP READER	1 = READ CHECK	1 = INOP	1 = MPE	1 = ANL	1 = MP	1 = BUSY	0 = DATA READY	1 = HOP EMPTY	0	0	1 = HOLD	1 = PICK	0	1 = ATT. READY

ANL = Array not loaded
MP = Multipunch
C0,C1 = Transmission Codes

00 = Transitional "Half-ASCII"
01 = Program loaded 8-bit data
10 = Binary 12 bit data
11 = Illegal

DMP LINE PRINTER

First Device Address 07_{16} DMP 7

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TRANSFER INITIATE		1	M	CONN. DI	CONN. SI	0	1= TERM ON NULL	1= PACK	1= ATT. INT.	1= DELETE FIRST CHAR.	1=VFU 0=LCNT	LINE COUNT OR VFU					
CONTROL	EOB/ TERM	0	1	IGNORE		EOB	TERM	0	MPE	0	MC	IGNORED					
	PAPER ADVANCE	0	1	CONN. DI	CONN. SI	0	0	1	1= ATT. INT.	0	1=VFU 0=LCNT	LINE COUNT OR VFU					
	NO-OP	0	1	CONN. DI	CONN. SI	0	0	0	1= ATT. INT.	0	0	0	0	0	0	0	0
SELECT COMMAND		0	0	0	0	0	0	0	TERMINATE CHARACTER								
STATUS		1= ERROR	0	0	1= INOP	1= MPE	1= DPC	1= DPO	1= BUSY	0= DATA READY	1= PAPER LOW	1= BOF	0	1= HOLD	1= NPO	0	1= ATT. INT.

DPC = DMP Controller DPO = Data Products Printer NPO = Paper Advance only occurred

ELECTROSTATIC PRINTER/PLOTTER (DEVICE ADDRESS 08_{16})

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TRANSFER INITIATE		1	M	CONN. DI	CONN. SI	0	*PLOT	**SPP	IGNORED								
CONTROL	EOB/ TERM	0	1	IGNORED		EOB	TERM	IGN	MPE	IGNORED							
	NO-OP	0	1	CONN. DI	CONN. SI	0	0	0	IGNORED								
STATUS		0= ERROR	0	0	INOP	MPE	0	0	1= BUSY	0= DATA READY	PAPER LOW	0	0	0	0	0	0

*PLOT: 1 = PLOT
0 = PRINT

**SPP: 1 = SIMULTANEOUS PRINT/PLOT
0 = NORMAL

X-Y PLOTTER (DEVICE ADDRESS 08_{16})

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TRANSFER INITIATE		1	0	CONN. DI	CONN. SI	0	IGNORED										
CONTROL	EOB/ TERM	0	1	IGNORED		1= EOB	1= TERM	IGNORED									
	OUTPUT DATA	IGNORED										1=PEN DOWN	1=PEN UP	1=DRUM DOWN	1=DRUM UP	1=CARR. RIGHT	1=CARR. LEFT
												**		**		**	
	NO-OP	0	1	CONN. DI	CONN. SI	0	0	0	IGNORED								
STATUS		0= ERROR	0	0	1= IN OP	0	0	0	1= BUSY	0= DATA	0	0	0	0	0	0	0

**MUTUALLY EXCLUSIVE

PAPER TAPE PUNCH (DEVICE ADDRESS 09_{16})

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TRANSFER INITIATE		1	0	CONN. DI	CONN. SI	0	IGNORED										
CONTROL	EOB/ TERM	0	1	IGNORED		EOB	TERM.	IGN	1= ABORT	IGNORED							
	NO-OP	0	1	CONN. DI	CONN. SI	0	0	0	IGNORED								
STATUS		1	0	0	0	0	0	0	1= BUSY	0= DATA READY	1= TAPE LOW	0	0	0	0	0	0

CONSOLE TTY/PAPER TAPE READER (DEVICE ADDRESS $0A_{16}$)

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TRANSFER INITIATE		1	0	CONN. DI	CONN. SI	1=IN 0=OUT	1= KEY BD.	ENABLE CLOCK	CLEAR BUFFER	IGNORED				0=REV.* 1=FOR.	IGNORED		
CONTROL	EOB/ TERM	0	1	IGNORED		0	TERM	IGN	1=ABORT	' IGNORED							
	NO-OP	0	1	CONN. DI	CONN. SI	0	0	0	IGNORED								
STATUS		1	0	0	0	0	0	0	1= BUSY	0= DATA READY	0	0	0	0	0	0	0

*NOTE: HIGH SPEED READER NORMALLY OPERATES IN REVERSE MODE.

HIGH LEVEL AIS OUTPUT UNIT (DEVICE ADDRESS 10_{16})

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TRANSFER INITIATE		1	M	CONN. DI	IGN	1=SEQ 0=RAN	IGNORED				*END ADDRESS						
CONTROL	EOB/ TERM	0	1	IGNORED		EOB	TERM	IGNORED			*START ADDRESS						
	NO-OP	0	1	CONN. DI	IGN	0	0	0	IGNORED								
STATUS		ERROR	IGNORED						1= BUSY	0= DATA READY	IGNORED						
DATA WORD		IGNORED									* CHANNEL ADDRESS						

* ADDRESSES ARE IN BINARY (0-127)

LINE PRINTER 50-150 LPM (DEVICE ADDRESS $0B_{16}$)

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TRANSFER INITIATE		1	0	CONN. DI	CONN. SI	0	IGNORED										
CONTROL	EOB/ TERM	0	1	IGNORED		EOB	TERM	IGNORED									
	NO-OP	0	1	CONN. DI	CONN. SI	0	0	0	IGNORED								
STATUS		0= ERROR	0	0	1= IN OP	0	0	0	1= CONT. BUSY	0= DATA READY	1= LOW PAPER	1= BOTTOM OF FORM	0	1= HOLD	0	0	0

HIGH LEVEL AIS INPUT UNIT (DEVICE ADDRESS 11_{16})

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TRANSFER INITIATE		1	M	CONN. DI	CONN. SI	IGNORED											
CONTROL	EOB/ TERM	0	1	IGNORED		EOB	TERM	IGNORED									
	NO-OP	0	1	CONN. DI	CONN. SI	0	0	0	IGNORED								
STATUS		0= ERROR	IGNORED						1= BUSY	0= DATA READY	IGNORED						
DATA WORD		SIGN	1	0	1	1	2^{10}	2^9	2^8	2^7	2^6	DATA 2^5	2^4	2^3	2^2	2^1	2^0

WIDE RANGE SOLID STATE AIS CUTPUT UNIT (DEVICE ADDRESS 12_{16})

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TRANSFER INITIATE		1	M	CONN. DI	IGNORED		1= RE-CYCLE	IGNORED									
CONTROL	EOB/ TERM	0	1	IGNORED		EOB	TERM	IGNORED									
	NO-OP	0	1	CONN. DI	IGN	0	0	0	IGNORED								
STATUS		ERROR	IGNORED						1= BUSY	0= DATA READY	IGNORED						
DATA WORD 1		IGN	GAIN 2^3	2^2	2^1	2^0	1= ZERO SUPPR.	1= AUTO	2^8	2^7	* CHANNEL ADDRESS $(0-127)_2$ 2^6	2^5	2^4	2^3	2^2	2^1	2^0
SUPPRESSION WORD		SIGN	IGN	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

* ADDRESSES ARE IN BINARY (0-127)

WIDE RANGE SOLID STATE AIS INPUT UNIT (DEVICE ADDRESS 13_{16})

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TRANSFER INITIATE		1	M	CONN. DI	CONN. SI	IGNORED											
CONTROL	EOB/ TERM	0	1	IGNORED		EOB	TERM	IGNORED									
	NO-OP	0	1	CONN. DI	CONN. SI	0	0	0	IGNORED								
STATUS		ERROR	IGNORED						BUSY	DATA READY	IGNORED						
DATA WORD		SIGN	GAIN 2^3	2^2	2^1	2^0	DATA 2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

WIDE RANGE RELAY AIS (DEVICE ADDRESS 14_{16})

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TRANSFER INITIATE		1	0	CONN. DI	CONN. SI	IGNORED											
CONTROL	EOB/ TERM	0	1	IGNORED		EOB	TERM	IGNORED									
	NO-OP	0	1	CONN. DI	CONN. SI	0	0	0	IGNORED								
STATUS		ERROR	IGNORED						BUSY	INPUT DATA READY	IGNORED						OUTPUT DATA READY
DATA WORD 1		1=RES. CHECK MODE	GAIN				1= ZERO SUPPR.	1= AUTO	* CHANNEL ADDRESS								
SUPPRESSION WORD		SIGN	IGN.	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

* ADDRESSES ARE IN BINARY (0-511)

I/OIS#1 (DEVICE ADDRESS 20-$2F_{16}$) I/OIS#2 (DEVICE ADDRESS 30-$3F_{16}$)

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CONTROL	I/O INTERRUPT COUPLER	0	1	CONN. DI	CONN. SI	IGNORED			1=RES REQ.	INTERRUPT CHANNEL 0	1	2	3	4	5	6	7
	INTERVAL TIMER	0	1	IGNORED							0=SINGLE 1=RECYCLE	IGNORED					
	CHANNEL MULTIPLEXER	0	1	IGNORED								EXPANDER ADDRESS			CHANNEL ADDRESS		
	SACI	0	1	IGNORED		0	T	IGNORED									
	SACI NO-OP	0	1	CONN. DI	CONN. SI	0	0	0	IGNORED								
TRANSFER INITIATE	SACI	1	0	CONN. DI	CONN. SI	1= IN	1= KEY BD.	IGNORED									
STATUS	SACI	ERROR	IGNORED		1=DATA SET NOT READY	IGNORED			BUSY	DATA READY	IGNORED						
	SYNCHRONIZER	SYS. CONN.	IGNORED							1= XFER REQ.	IGNORED						
	SUBSYSTEM	SYS CONN.	IGNORED														
DATA	ANALOG OUTPUT	SIGN	0=CH1 1=CH2	CRT CONT.	CRT CONT.	CRT CONT.	DATA 2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
	SACI	IGNORED								ASCII DATA							

DISC/CONSOLE

	BIT	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
TRANSFER INITIATE		1	0	CONN DI	CONN SI	1 IN 0 OUT	1- KEY BD	0	CLEAR BUF-FER	0	0- CH/LF DELAY	0	' 0 ECHO	0 REV 1-FOR	0	0	0
CONTROL	TERM	0	1	0	0	0	TERM	0	1 ABORT	0	ICB	0	0	0	0	0	0
	NO-OP	0	1	CONN DI	CONN SI	0	0	0	0	0	0	0	0	0	0	0	0
	BREAK	0	1	CONN DI	CONN SI	0	0	0	1	0	0	1 BRK DET	0	0	0	0	0
	SI REL	0	1	X	X	0	0	0	1	0	1	0	0	0	0	0	0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
STATUS	1	0	0	0	0	0	0	1 BUSY	0- DATA READY	INPUT READY	1- BREAK	FULL DUPLEX	0	0	0	0

ASYNCHRONOUS TERMINAL CONTROLLER

COMMAND INSTRUCTION CONTROLLER 41Ra8

BIT NUMBER	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CHANNEL SELECT	0	1	DI	SI		0	0					CHANNEL ADDRESS				
RELEASE	1	0	0	0		0	1									
MASTER CLEAR	1	0	0	0		1	0									

COMMAND INSTRUCTION CHANNEL 41Ra9

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LOAD PARAMETERS	1	1			PARITY		FRAME SIZE		STOP BITS				BAUD RATE			
INITIATE	1	0	DI	SI				QUICK INIT								
MODE	0	1	DI	SI		TERM	ABORT	POL + −				ECHO / WA			RTS	
SERVICE INTERRUPT	0	0			1											

DATA INSTRUCTION 45Ra9,4DRa9

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OUTPUT 45Ra9									DATA BYTE							
INPUT 4DRa9	ERROR								DATA BYTE							

INPUT STATUS INSTRUCTION 49Ra8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CONTROLLER	PRESENT	BUSY	DIIS	SIIS								CHANNEL ADDRESS				

INPUT STATUS INSTRUCTION CHANNEL 49Ra9

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
OUTPUT CHNL	1						TERM PEND	BUSY	DATA READY	C	0	0	0	0	CTS	0
INPUT CHNL	ERROR	BREAK	PARITY	0	OVER RUN	0	0	BUSY	DATA READY	FRAME ERROR	0	0	0	INPUT DATA	CDET	0

Appendix D. Part 1. INSTRUCTION SET - ALPHABETICAL REFERENCE

				Execution Time (Microseconds)		
Mnemonic	*Name*	*Notes*	*OP Code*	*No Branch*	*Branch*	*Page*
ABMB	Add Bit in Memory and Branch if Nonzero	2,3	84	1.5	2.2	9-22
ABMM	Add Bit in Memory	2	80	1.1		9-22
ABR	Add Bit in Register		60	.2		9-19
ABRB	Add Bit in Register and Branch if Nonzero	3	70	.6	1.3	9-19
ABSB	Add Bit in Memory (Short-Displaced) and Branch if Nonzero	2,3	94	1.5	2.2	9-23 9-
ABSM	Add Bit in Memory (Short-Displaced)	2	90	1.1		9-23
ABXB	Add Bit in Memory (Short-Indexed) and Branch if Nonzero	2,3	9C	1.5	2.2	9-23
ABXM	Add Bit in Memory (Short-Indexed)	2	98	1.1		9-23
ADES	Add Immediate with Extended Sign		E8X2	.6		9-38
ADI	Add Memory (Immediate) to Register		E8X0	.4		9-20
ADM	Add Memory to Register		E0	1.1		9-20
ADMB	Add Register to Memory if Nonzero	2,3	C4	1.5	2.2	9-24
ADMD	Add Memory Doubleword to Double-Register		C8	1.5		9-21
ADMM	Add Register to Memory	2	C0	1.1		9-24
ADR	Add Register to Register		68	.2		9-19
ADRB	Add Register to Register and Branch if Nonzero	3	78	.6	1.3	9-19
ADRD(DAR)	Add Double-Register to Double-Register		C8	.4		9-20
ADS	Add Memory (Short-Displaced) to Register		F0	1.1		9-20
ADSB	Add Register to Memory (Short-Displaced) and Branch if Nonzero		D4	1.5	2.2	9-24
ADSM	Add Register to Memory (Short-Displaced)		D0	1.1		9-24
ADX	Add Memory (Short-Indexed) to Register		F8	1.1		9-21
ADXB	Add Register to Memory (Short-Indexed) and Branch if Nonzero	2,3	DC	1.5	2.2	9-25
ADXM	Add Register to Memory (Short-Indexed)	2	D8	1.1		9-25
BLCR	Branch and Link on Condition Code C Reset	3	A7	.4	1.1	9-115
BLCS	Branch and Link on Condition Code C Set	3	A7	.4	1.1	9-115
BLGE	Branch and Link on Greater Than or Equal Condition	3	A7	.4	1.1	9-115
BLGT	Branch and Link on Greater Than Condition	3	A7	.4	1.1	9-116

				Execution Time (Microseconds)		
Mnemonic	*Name*	*Notes*	*Op Code*	*No Branch*	*Branch*	*Page*
BLHI	Branch and Link on Magnitude Higher Condition	3	A7	.4	1.1	9-116
BLI	Branch and Link (Immediate)		EF	.4		9-112
BLLE	Branch and Link on Less Than or Equal Condition	3	A7	.4	1.1	9-116
BLLS	Branch and Link on Less Than Condition	3	A7	.4	1.1	9-115
BLM	Branch and Link on Magnitude Not Higher Condition	1	E7		1.1	9-112
BLNH	Branch and Link on Magnitude Not Higher Condition	3	A7	.4	1.1	9-116
BLNR	Branch and Link on Condition Code N Reset	3	A7	.4	1.1	9-113
BLNS	Branch and Link on Condition Code N Set	3	A7	.4	1.1	9-113
BLOR	Branch and Link on Condition Code O Reset	3	A7	.4	1.1	9-114
BLOS	Branch and Link on Condition Code O Set	3	A7	.4	1.1	9-114
BLT	Branch and Link (Indexed Through-Table)	1	F7		2.0	9-112
BLX	Branch and Link (Short-Indexed)	1	FF		1.1	9-112
BLZR	Branch and Link on Condition Code Z Reset	3	A7	.4	1.1	9-114
BLZS	Branch and Link on Condition Code Z Set	3	A7	.4	1.1	9-114
BRM	Branch to Microroutine Immediate		0E	.4		9-143
BRMI	Branch to Microroutine Immediate		0E	.4		9-143
BRU	Branch Unconditionally	1	E7		1.1	9-103
BRX	Branch (Short-Indexed) Unconditionally	1	E7		1.1	9-103
BXCR	Branch (Short-Indexed) on Condition Code C Reset	3	8F	.4	1.1	9-110
BXCS	Branch (Short-Indexed) on Condition Code C Set	3	8F	.4	1.1	9-110
BXGE	Branch (Short-Indexed) on Greater Than or Equal Condition	3	8F	.4	1.1	9-110
BXGT	Branch (Short-Indexed) on Greater Than Condition	3	8F	.4	1.1	9-111
BXHI	Branch (Short-Indexed) on Magnitude Higher Condition	3	8F	.4	1.1	9-111
BXLE	Branch (Short-Indexed) on Less Than or Equal Condition	3	8F	.4	1.1	9-111
BXLS	Branch (Short-Indexed) on Less Than Condition	3	8F	.4	1.1	9-110
BXNH	Branch (Short-Indexed) on Magnitude Not Higher Condition	3	8F	.4	1.1	9-111
BXNR	Branch (Short-Indexed) on Condition Code N Reset	3	8F	.4	1.1	9-108
BXNS	Branch (Short-Indexed) on Condition Code N Set	3	8F	.4	1.1	9-108

				Execution Time (Microseconds)		
Mnemonic	*Name*	*Notes*	*Op Code*	*No Branch*	*Branch*	*Page*
BXOR	Branch (Short-indexed) On Condition Code O Reset	3	8F	.4	1.1	9-109
BXOS	Branch (Short-Indexed) On Condition Code O Set	3	8F	.4	1.1	9-109
BXZR	Branch (Short-Indexed) on Condition Code Z Reset	3	8F	.4	1.1	9-109
BXZS	Branch (Short-Indexed) on Condition Code Z Set	3	8F	.4	1.1	9-109
CAR	Clear Active and Return		24	7.3		9-151
CBMB	Compare Bit and Memory and Branch if Equal or Less	3	87	1.7	2.2 (=), 2.4 (<)	9-95
CBSB	Compare Bit and Memory (Short-Displaced) and Branch if Equal or Less	3	97	1.7	2.2 (=), 2.4 (<)	9-95
CBXB	Compare Bit and Memory (Short-Indexed) and Branch	3	9F	1.7	2.2 (=), 2.4 (<)	9-96
CDFI	Convert Double Precision Floating Point Operand to Double Integer		36	2.2		9-63
CDIF	Convert Double-Register Integer to Floating Point		30	1.3		9-63
CFDI	Convert Floating Point to Double-Register Integer		34	2.2		9-63
CIES	Compare Immediate with Extended Sign		E8X4	.6		9-39
CIR	Clear Interrupt and Return		25	7.5		9-151
CLM	Clear Memory	2	CE	.4		9-11
CLMD	Clear Memory Doubleword	2	CE	.6	2.2	9-11
CQFF	Convert Quad-Register Floating Point to Floating Point		35	1.2		9-63
CQFI	Convert Quad Precision Floating Point Operand to Double Integer		37	2.2		9-64
CRI	Compare Register with Memory (Immediate)		E9	.4		9-91
CRM	Compare Register with Memory		C3	1.1		9-91
CRMB	Compare Register with Memory and Branch Equal or Less	3	C7	1.7	2.2 (=), 2.4 (<)	9-91
CRMD	Compare Double-Register with Memory Doubleword		CF	1.5		9-94
CRR	Compare Register with Register		6E	.2		9-90
CRRD	Compare Double-Register with Double Register		CF	.4		9-90
CRRQ	Compare Quad Register to Quad Register		8B	.8		9-90
CRRT	Compare Triple Register to Triple Register		8B	.6		9-90

25

				Execution Time (Microseconds)		
Mnemonic	Name	Notes	Op Code	No Branch	Branch	Page
CRS	Compare Register with Memory		D3	1.1		9-91
CRSB	Compare Register with Memory (Short-Displaced) and Branch if Equal or Less	3	D7	1.7	2.2 (=), 2.4 (<)	9-92
CRX	Compare Register with Memory		DB	1.1		9-92
CRXB	Compare Register with Memory (Short-Indexed) and Branch if Equal or Less	3	DF	1.7	2.2 (=), 2.4 (<)	9-93
CRXD	Compare Double Register to Short-Indexed Memory Doubleword		8E	1.5		9-93
CRZ	Compare Register to Zero		E8	.2		9-94
CRZD	Compare Double Register to Zero		E8	.4		9-94
DMPI	Initialize Direct Memory Processor		05	3.1		9-161
DVES	Divide Immediate with Extended Sign		0E	9.2		9-39
DVI	Divide Immediate		E8	5.4		9-34
DVM	Divide Register by Memory		A1	6.1		9-34
DVMD	Divide Quad-Register by Memory Doubleword		A3	9.7		9-35
DVR	Divide Register by Register		21	5.0		9-33
DVRD	Divide Quad-Register by Double-Register		A3	8.4		9-33
DVS	Divide Register by Memory (Short-Displaced)		B1	6.1		9-34
DVX	Divide Register by Memory (Short-Indexed)		B9	6.1		9-34
EPMD	Enter Pipeline Mode of Execution	1	E8	1.1		9-128
ESD	Extend Sign Double		8C	1.2		9-37
ESS	Extend Sign Single		8C	.6		9-37
ETI	Extract Memory (Immediate) from Register		EA	.4		9-65
ETM	Extract Memory from Register		E2	1.1		9-65
ETMB	Extract Register from Memory and Branch if Nonzero	2,3	C5	1.5	2.2	9-67
ETMM	Extract Register from Memory	2	C1	1.1		9-67
ETR	Extract Register from Register		6A	.2		9-65
ETRB	Extract Register from Register and Branch if Nonzero		7A	.6		9-65
ETS	Extract Memory (Short-Displaced) from Register		F2	1.1		9-66
ETSB	Extract Register from Memory (Short-Displaced) and Branch if Nonzero	2,3	D5	1.5	2.2	9-68
ETSM	Extract Register from Memory (Short-Displaced)	2	D1	1.1		9-68
ETX	Extract Memory (Short-Indexed) from Register		FA	1.1		9-66

				Execution Time (Microseconds)		
Mnemonic	*Name*	*Notes*	*Op Code*	*No Branch*	*Branch*	*Page*
ETXB	Extract Register from Memory Short-Indexed and Branch if Nonzero	2,3	DD	1.5	2.2	9-68
ETXM	Extract Register from Memory (Short-Indexed)	2	D9	1.1		9-68
EVMO	Enter Virtual Mode of CPU Execution	1	01	1.1		9-127
EXI	Execute Immediate		E8	.4		9-119
EXMA	Enter Extended Memory Addressing Mode		01	.2		9-125
EXR	Execute Instruction Word in Register(s)		B3	.4		9-119
FAI	Floating Point Add Immediate		38	1.5		9-45
FAID	Floating Point Add Immediate Double Precision		3C	1.7		9-46
FAIQ	Floating Point Add Immediate Quad Precision		3C	1.9		9-47
FAM	Floating Point Add Memory Doubleword to Register		38	2.4		9-44
FAMD	Floating Point Add Memory Tripleword to Triple-Register		3C	2.8		9-45
FAMQ	Floating Point Add Memory Quadword to Quad-Register		3C	3.0		9-46
FAR	Floating Point Add Double-Register to Double Register		30	1.2		9-43
FARD	Floating Point Add Triple Register to Triple Register		43	1.4		9-43
FARQ	Floating Point Add Quad Register to Quad Register		43	1.6		9-44
FDI	Floating Point Divide Immedaite		33	6.3		9-60
FDID	Floating Point Divide Immediate Double Precision		3F	9.7		9-61
FDIQ	Floating Point Divide Immediate Quad Precision		3F	13.1		9-62
FDM	Floating Point Divide Double-Register by Memory Doubleword		3B	7.4		9-59
FDMD	Floating Point Divide Triple-Register by Memory Tripleword		3F	11.0		9-60
FDMQ	Floating Point Divide Quad-Register by Memory Quadword		3F	14.4		9-61
FDR	Floating Point Divide Double-Register by Double-Register		33	6.1		9-58
FDRD	Floating Point Divide Triple-Register by Triple-Register		37	9.5		9-58
FDRQ	Floating Point Divide Quad-Register by Quad-Register		37	12.9		9-59
FMI	Floating-Point Multiply Immediate		3A	2.0		9-55

25

				Execution Time (Microseconds)		
Mnemonic	*Name*	*Notes*	*Op Code*	*No Branch*	*Branch*	*Page*
FMID	Floating-Point Multiply Immediate Double Precision		3E	2.6		9-56
FMIQ	Floating-Point Multiply Immediate Quad Precision		3E	3.2		9-57
FMM	Floating-Point Multiply Double-Register by Memory Doubleword		3A	3.1		9-54
FMMD	Floating-Point Multiply Triple-Register by Memory Tripleword		3E	3.7		9-55
FMMQ	Floating-Point Multiply Quad Register by Memory Quadword		3E	4.3		9-56
FMR	Floating-Point Multiply Double-Register		32	1.8		9-53
FMRD	Floating Point Multiply Triple Register by Triple Register		36	2.4		9-53
FMRQ	Floating-Point Multiply Quad-Register by Quad-Register		36	3.0		9-54
FSI	Floating-Point Subtract Immedaite		39	1.5		9-50
FSID	Floating Point Subtract Immediate Double Precision		3D	1.7		9-51
FSIQ	Floating Point Subtract Immediate Quad Precision		3D	1.9		9-52
FSM	Floating-Point Subtract Memory Doubleword from Double-Register		39	2.4		9-49
FSMD	Floating Point Subtract Memory Tripleword from Triple-Register		3D	2.8		9-50
FSMQ	Floating Point Subtract Memory Quadword from Quad-Register		3D	3.0		9-51
FSR	Loating Point Subtract Double-Register from Double-Register		31	1.2		9-48
FSRD	Floating Point Subtract Triple-Register from Triple-Register		35	1.4		9-48
FSRQ	Floating-Point Subtract Quad-Register from Quad-Register		35	1.6		9-49
GMR	Generate Mask in Register (Load Negative Power of Two)		67	.8		9-4
GMRB	Generate Mask in Register and Branch Unconditionally		77		1.1	9-4
HCR	Hop on Condition Code C Reset	3	AB	.4	1.3	9-106
HCS	Hop on Condition Code C Set	3	AB	.4	1.3	9-106
HGE	Hop on Greater than or Equal Condition	3	AC	.4	1.3	9-107
HGT	Hop on Greater Than Condition	3	AD	.4	1.3	9-107
HHI	Hop on Magnitude Higher Condition	3	AG	.4	1.3	9-108
HLE	Hop on Less Than or Equal Condition	3	AD	.4	1.3	9-107
HLS	Hop on Less Than Condition	3	AC	.4	1.3	9-107

26

				Execution Time (Microseconds)		
Mnemonic	*Name*	*Notes*	*Op Code*	*No Branch*	*Branch*	*Page*
HLT	Halt (Privileged)	1				9-122
HNH	Hop on Magnitude Not Higher Condition	3	A6	.4	1.3	9-108
HNR	Hop on Condition Code N Reset	3	A8	.4	1.3	9-105
HNS	Hop on Condition Code N Set	3	A8	.4	1.3	9-105
HOP	Hop Unconditionally		F7		1.3	9-103
HOR	Hop on Condition Code O Reset	3	AA	.4	1.3	9-106
HOS	Hop on Condition Code O Set	3	AA	.4	1.3	9-106
HZR	Hop on Condition Code Z Reset	3	A9	.4	1.3	9-105
HZS	Hop on Condition Code Z Set		A9	.4	1.3	9-105
IBR	Interchange Bytes Register to Register		0A	.3		9-5.1
IDA,B,C,D	Input Data from I/O Group A,B,C, or D		4C, 4D, 4E, 4F	1.1 min	15.0 max	9-156
IRM	Interchange Register and Memory	2	B6	1.1		9-6
IRR	Interchange Register and Register		B7	.6		9-6
IRRD	Interchange Double Register and Double Register		8A	1.2		9-6
ISA,B,C,D	Input Status from I/O Group A,B,C, or D		48, 49 4A, 4B	1.1 min	15.0 max	9-155
ISZ	Input Status from Device Zero		48	1.1 min		9-154
LAD	Shift Left Arithmetic Double-Register		2E	.8 (<7)	1.2 (≤7)	9-86
LAQ	Shift Left Arithmetic Quadruple-Register		2E	1.2 (<7)	1.6 (≤7)	9-86
LAS	Shift Left Arithmetic Single-Register		2F	.6 (<7)	1.0 (≤7)	9-85
LBR	Load Bit in Register		65	.2		9-4
LBRB	Load Bit in Register and Branch Unconditionally		75		1.1	9-4
LBX	Load Byte from Memory (Byte-Indexed)		AE	1.3		9-7
LCCC	Load Register with Current Condition Code of PSD		0E	.4		9-120
LCIA	Load Register with Current Interrupt Active Latches		0E	.2		9-148
LCIE	Load Register with Current Interrupt Enable Latches		0E	.2		9-148
LCIR	Load Register with Current Interrupt Request Latches		0E	1.0		9-148
LCPR	Load Register with Current Program Register of PSD		0E	.2		9-120
LCPS	Load Register with Current Program Status Register of PSD		0E	.2		9-120
LDAM	Load Register from (Actual) Memory		BE	2.1		9-142
LDES	Load Immediate and Extend Sign		E8X1	.6		9-38
LDF	Load Immediate Floating-Point		ED	.4		9-42

31 (+6)

				Execution Time (Microseconds)		
Mnemonic	Name	Notes	Op Code	No Branch	Branch	Page
LDFD	Load Immediate Floating Point Double Precision		ED	.6		9-42
LDFQ	Load Immediate Floating Point Quad Precision		ED	.8		9-42
LDI	Load Register from Memory (Immediate)		ED	.4		9-8
LDM	Load Register from Memory		E5	1.1		9-7
LDMD	Load Double-Register from Memory Doubleword		CD	1.3		9-8
LDMT	Load Triple-Register from Memory Tripleword		88	1.5		9-9
LDS	Load Register from Memory		F5	1.1		9-8
LDVM	Load Register from Memory (Via Map Image)		BE	2.2		9-139
LDX	Load Register from Memory (Short-Indexed)		FD	1.1		9-8
LDXD	Load Double-Register from Memory Doubleword (Short-Indexed)		8D	1.3		9-9
LDXT	Load Triple-Register from Memory Tripleword (Short-Indexed)		88	1.5		9-9
LFM	Load File from Memory		A4	1.1 plus 0.2 per transfer		9-9
LFS	Load File from Memory (Short-Displaced)		B4	1.1 plus 0.2 per transfer		9-10
LFX	Load File from Memory (Short-Indexed)		BC	1.1 plus 0.2 per transfer		9-10
LIMP	Load Instruction Map Image into Hardware Map		01	5.3 per 4 transfer		9-132
LLD	Shift Left Logical Double-Register		2C	.8 (<7)	1.2 (≤7)	9-88
LLQ	Shift Left Logical Quadruple-Register		2C	1.2 (<7)	1.6 (≤7)	9-88
LLS	Shift Left Logical Single-Register		2D	.6 (<7)	1.0 (≤7)	9-87
LOMP	Load Operand Map Image Into Hardware Map		01	4.9 per 4 transfers		9-132
LRS	Left Rotate Single Register to Register		FF	.4		9-89
LTDD	Loop Termination with Directly Addressed Control Variable and Directly Addressed Terminal Value		E8	2.9	1.9	9-123
LTDL	Loop Termination with Directly Addressed Control Variable and Literal Terminal Value		E8	2.9	1.9	9-123
LTID	Loop Termination with Indirectly Addressed Control Variable and Directly Addressed Terminal Value		E8	3.8	2.8	9-124
LTIL	Loop Termination with Indirectly Addressed Control Variable and Literal Terminal Value		E8	3.8	2.8	9-124
LXR	Load Extended Memory Control Register		02	.2		9-126
MBEV	Move Block from Extended to Virtual Memory		0E	2.5 per 4 transfers		9-17
MBL	Move Byte Left Register to Register		09	.3		9-5

27

				Execution Time (Microseconds)		
Mnemonic	*Name*	*Notes*	*Op Code*	*No Branch*	*Branch*	*Page*
MBR	Move Byte Right Register to Register		08	.3		9-5
MBVE	Move Block from Virtual to Extended Memory		0E	2.5 per 4 transfers		9-17
MBVV	Move Virtual Block to Virtual Block		0E	2.3 per 4 transfers		9-18
MLR	Move Lower Byte Register to Register		0C	.3		9-5
MMRB	Move Memory File to Register Block Section of Context File		06	4.5		9-129
MPES	Multiply Immediate with Extended Sign		0E	2.4		9-39
MPI	Multiply Immediate		E8	1.8		9-31
MPM	Multiply Register by Memory		A0	2.7		9-31
MPMD	Multiply Double-Register by Memory Doubleword		A2	3.3		9-32
MPR	Multiply Register by Register		20	1.4		9-31
MPRD	Multiply Double-Register by Double-Register		A2	1.8		9-31
MPS	Multiply Register by Memory (Short-Displaced)		B0	2.7		9-32
MPX	Multiply Register by Memory (Short-Indexed)		B8	2.7		9-32
MRBM	Move Register Block Section of Context File to Memory File		07	8.3		9-129
MUR	Move Upper Byte Register to Register		0B	.3		9-5
NOP	No Operation					9-118
OBMM	OR Bit in Memory	2	82	1.1		9-72
OBR	OR Bit in Register		63	.4		9-69
OBRB	OR Bit in Register and Branch Unconditionally	1	73		1.1	9-69
OBSM	OR Bit in Memory (Short-Displaced)	2	92	1.1		9-72
OBXM	OR Bit in Memory (Short-Indexed)	2	9A	1.1		9-73
OCA,B,C,D	Output Command to I/O Group A, B, C, or D		40, 41, 42, 43	1.9 min	15.0 max	9-158
ODA,B,C,D	Output Data to I/O Group A, B, C, or D		44,45,46,47	1.9 min	15.0 max	9-159
ORI	OR Memory (Immediate) to Register		E3	.4		9-70
ORM	OR Memory to Register		E3	1.1		9-70
ORMM	OR Register to Memory	2	C2	1.1		9-73
ORR	OR Register to Register		6B	.2		9-70
ORRB	OR Register to Register and Branch if Nonzero	3	7B	.6	1.3	9-70
ORS	OR Memory (Short-Displaced) to Register		F3	1.1		9-71
ORSM	OR Register to Memory (Short-Displaced	2	D2	1.1		9-73
ORX	OR Memory (Short-Indexed) to Register		FB	1.1		9-71
ORXM	OR Register to Memory (Short-Indexed)		DA	1.1		9-73

32 (+6)

				Execution Time (Microseconds)		
Mnemonic	Name	Notes	Op Code	No Branch	Branch	Page
PLM	Pull Register(s) from Memory LIFO Stock		BA	3.5 + 0.2 per transfer		9-16
PSM	Pull Register(s) into Memory LIFO Stock		BB	2.6 + 0.2 per transfer		9-15
RAD	Shift Right Arithmetic Double-Register		2A	.7 (<8)	.9 (≥8)	9-86
RAQ	Shift Right Arithmetic Quadruple-Register		2A	1.1 (<8)	1.3 (≥8)	9-86
RAS	Shift Right Arithmetic Single-Register		2B	.5 (<8)	.7 (≥8)	9-85
RDI	Read Internal Registers		0E	.8		9-144
REX	Request Executive Service		23	1.4		9-147
RIA	Reset Interrupt Active		27	1.2		9-150
RIE	Reset Interrupt Enable		27	1.2		9-150
RIR	Reset Interrupt Request		27	1.2		9-150
RLD	Shift Right Logical Double-Register		28	.7 (<8)	.7 (≥8)	9-88
RLQ	Shift Right Logical Quadruple-Register		28	1.1 (<8)	1.3 (≥8)	9-88
RLS	Shift Right Logical Single-Register		29	.5 (<8)	.7 (≥8)	9-87
RMI	Request Multiprocessor Interrupt		01	.5		9-152
RMPS	Read Memory Plane Status		03	1.3		9-145
RMWS	Read Memory Word Status		03	1.3		9-145
SBR	Subtract Bit in Register		61	.2		9-26
SBRB	Subtract Bit in Register and Branch if Nonzero		71	.6	1.3	9-26
SBX	Store Byte in Memory (Byte-Indexed)	3	AF	1.0		9-12
SCCC	Select Current Condition Codes in PSD		B2	.2		9-121
SCRB	Select Current Register Block in PSD		01	6.2		9-130
SFM	Store File in Memory	2	A5	.4 plus 0.2 per transfer		9-14
SFS	Store File in Memory (Short-Displaced)	2	B5	.4 plus 0.2 per transfer		9-14
SFX	Store File in Memory (Short-Indexed)	2	BD	.4 plus 0.2 per transfer		9-14
SIA	Set Interrupt Active		26	1.2		9-149
SIE	Set Interrupt Enable		26	1.2		9-149
SIOM	Select Another Program's IM as Current OM		01	1.6		9-137
SIR	Set Interrupt Request		26	1.2		9-149
SOMP	Store Operand Map into Map Image	1	01	2.0 per transfer		9-133
SOOM	Select Another Program's OM as Current OM		01	1.4		9-137
SRCR	Set Register if Condition Code C Reset		CA	.4		9-81
SRCS	Set Register if Code C Set		CA	.4		9-81
SRGE	Set Register on Greater than or Equal Condition		CA	.4		9-82
SRGT	Set Register on Greater than Condition		CA	.4		9-82
SRHI	Set Register on Magnitude Higher Condition		CA	.4		9-83
SRLE	Set Register on Less than or Equal Condition		CA	.4		9-82
SRLS	Set Register on Less than Condition		CA	.4		9-82

				Execution Time (Microseconds)		
Mnemonic	*Name*	*Notes*	*Op Code*	*No Branch*	*Branch*	*Page*
SRNH	Set Register on Magnitude not Higher Condition		CA	.4		9-83
SRNR	Set Register if Condition Code N Reset		CA	.4		9-80
SRNS	Set Register if Condition Code N Set		CA	.4		9-80
SROR	Set Register if Condition Code O Reset		CA	.4		9-81
SROS	Set Register if Condition Code O Set		CA	.4		9-81
SRZR	Set Register if Condition Code Z Reset		CA	.4		9-80
SRZS	Set Register if Condition Code Z Set		CA	.4		9-80
STAM	Store Register into (Actual) Memory	2	AF	1.4		9-142
STI	Store Register in Memory (Immediate)		EE	.6		9-12
STM	Store Register in Memory	2	E6	.4		9-12
STMD	Store Double-Register into Memory Double-Word	2	CE	.6		9-13
STMT	Store Triple-Register into Memory Triple-Word	2	88	.8		9-13
STS	Store Register into Memory (Short-Displaced)	2	F6	.4		9-12
STVM	Store Register into Memory (Via Map Image)	2	AF	1.5		9-139
STX	Store Register in Memory (Short-Indexed)	2	FE	.4		9-13
STXD	Store Double-Register into Memory Doubleword (Short-Indexed)	2	8E	.6		9-13
STXT	Store Triple-Register into Memory Tripleword (Short-Indexed)	2	88	.8		9-14
SUES	Subtract Immediate with Extended Sign		E8X3	.6		9-39
SUI	Subtract Memory (Immediate) from Register		E9	.4		9-28
SUM	Subtract Memory from Register		E1	1.1		9-27
SUMD	Subtract Memory Doubleword from Double-Register		C9	1.5		9-28
SUR	Subtract Register from Register		69	.2		9-27
SURB	Subtract Register from Register and Branch if Nonzero	3	79	.6	1.3	9-27
SURD	Subtract Double-Register from Double-Register		C9	.4		9-27
SUS	Subtract Memory (Short-Displaced) from Register		F1	1.1		9-28
SUX	Subtract Memory (Short-Indexed) from Register		F9	1.1		9-28
SZOM	Select Map Zero as Current OM		01	1.2		9-137
TBMB	Test Bit(s) in Memory and Branch if One	3	86	1.5	2.2	9-99
TBMM	Test Bit(s) in Memory		83	1.1		9-99
TBR	Test Bit(s) in Register		66	.2		9-97

				Execution Time (Microseconds)		
Mnemonic	*Name*	*Notes*	*Op Code*	*No Branch*	*Branch*	*Page*
TBRB	Test Bit in Register and Branch if One	3	76	.6		9-97
TBSB	Test Bit(s) in Memory (Short-Displaced) and Branch if One	3	96	1.5		9-100
TBSM	Test Bit(s) in Memory (Short-Displaced)		93	1.1		9-100
TBXB	Test Bit in Memory (Short-Indexed) and Branch if One	3	9E	1.5		9-100
TBXM	Test Bit(s) in Memory (Short-Indexed)		9B	1.1		9-100
TERB	Test Register and Branch if any Ones Compare	3	7E	.6	1.3	9-98
TETI	Test Extract Memory (Immediate) from Register		EA	.4		9-66
TOR	Transfer One's Complement Register to Register		0D	.2		9-79
TORI	Test OR Memory (Immediate) to Register		EB	.4		9-71
TRMB	Test Register and Memory and Branch if any Ones Compare	3	C6	1.5	2.2	9-98
TRO	Transfer and Reset Overflow Status History		0E	.6		9-120
TRR	Transfer Register to Register		6D	.2		9-5.1
TRRB	Transfer Register to Register and Branch if Nonzero	3	7D	.6	1.3	9-5.1
TRRD	Transfer Double-Register to Double-Register		CD	.4		9-5.1
TRRQ	Transfer Quadruple-Register to Quadruple Register		8D	.8		9-5.1
TRSB	Test Register and Memory (Short- Displaced)		D6	1.5		9-98
TRXB	Test Register and Memory (Short-Indexed) and Branch if any Ones Compare	3	DE	1.5	2.2	9-98
TSBM	Test and Set Bit Memory	2				9-99
TTR	Transfer Two's Complement of Register to Register		6F	.2		9-36
TTRB	Transfer Two's Complement of Register to Register and Branch if Nonzero	3	7F	.6	1.3	9-36
TTRD	Transfer Two's Complement of Double-Register to Double-Register		8A	.4		9-36
TTRQ	Transfer Two's Complement of Quadruple-Register to Quadruple-Register		8B	.8		9-36
TXOI	Test Exclusive OR Memory (Immediate) to Register		EC	.4		9-76
WRI	Write Internal Register		0E	.7		9-144
WMS	Write Memory Status		04	.6		9-146
XBR	Exclusive OR Bit in Register		64	.4		9-74

26

				Execution Time (Microseconds)		
Mnemonic	*Name*	*Notes*	*Op Code*	*No Branch*	*Branch*	*Page*
XBRB	Exclusive OR Bit in Register and Branch if Nonzero	3	74	.8	1.5	9-74
XOI	Exclusive OR Memory to Register (Immediate)		EC	.4		9-75
XOM	Exclusive OR Memory to Register		E4	1.1		9-75
XOR	Exclusive OR Register to Register		6C	.2		9-75
XORB	Exclusive OR Register to Register and Branch if Nonzero	3	7C	.6	1.3	9-75
XOS	Exclusive OR Memory to Register (Short-Displaced)		F4	1.1		9-76
XOX	Exclusive OR Memory to Register (Short-Displaced)		FC	1.1		9-76
XPMD	Exit Pipeline Mode of Execution	1	E8	1.1		9-128
XVMO	Exit Virtual Mode of CPU Execution	1	01	1.1		9-127
ZBMB	Zero Bit in Memory and Branch if Nonzero	2,3	85	1.5	2.2	9-77
ZBMM	Zero Bit in Memory	2	81	1.1		9-77
ZBR	Zero Bit in Register		62	.4		9-77
ZBRB	Zero Bit in Register and Branch if Nonzero	3	72	.8	1.5	9-77
ZBSB	Zero Bit in Memory (Short-Displaced) and Branch	2,3	95	1.5	2.2	9-78
ZBSM	Zero Bit in Memory (Short-Displaced)	2	91	1.1		9-78
ZBXB	Zero Bit in Memory (Short-Indexed) and Branch if Nonzero	2,3	9D	1.5	2.2	9-78
ZBXM	Zero Bit in Memory (Short-Indexed)	2	93	1.1		9-78
ZIMP	Zero Section of Instruction Map	1	01	10.4 per 16 transfers		9-135
ZOMP	Zero Section of Operand Map	1	01	10.0 per 16 transfers		9-135

19

Reference Notes:

1. Execution of this instruction always results in a CPU memory interface flush.
2. Execution of this instruction will always result in a CPU memory interface flush when in the non-pipelined mode of CPU operation.
3. Execution of this instruction will result in a CPU memory interface flush when the conditional HOP or BRANCH is taken.

Part 2. CLASSIC INSTRUCTION MAP

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	HLT	AUG 01	LXR	RMPS RMWS	WMS	DMPI	MMRB	MRBM	MBR	MBL	IBR	MUR	MLR	TOR	AUG 0E	LRS
1						RESERVED FOR DECIMAL ARITHMETIC										
2	MPR	DVR	DAR	REX	CAR	CIR	SIA SIE SIR	RIA RIE RIR	RLD RLQ	RLS	RAD RAQ	RAS	LLD LLQ	LLS	LAD LAQ	LAS
3	FAR CDIF	FSR	FMR	FDR	FARD FARQ CFDI	FSRD FSRQ CQFF	FMRD FMRQ CDFI	FDRD FDRQ CQFI	FAM FAI	FSM FSI	FMM FMI	FDM FDI	FAMD FAMQ FAID FAIQ	FSMD FSMQ FSID FSIQ	FMMD FMMQ FMID FMIQ	FDMD FDMQ FDID FDIQ
4	OCA	OCB	OCC	OCD	ODA	ODB	ODC	ODD	ISA	ISB	ISC	ISD	IDA	IDB	IDC	IDD
5						RESERVED FOR COMMUNICATIONS										
6	ABR	SBR	ZBR	OBR	XBR	LBR	TBR	GMR	ADR	SUR	ETR	ORR	XOR	TRR	CRR	TTR
7	ABRB	SBRB	ZBRB	OBRB	XBRB	LBRB	TBRB	GMRB	ADRB	SURB	ETRB	ORRB	XORB	TRRB	TERB	TTRB

AUGMENTS

Ra	01	Ra	01	Rb	0E	Rb	0E
0	RMI	8	XVMO	0	TRO	8	MBVE
1	EVMO	9	ZIMP	1	LCPS	9	MBEV
2	SIOM	A	UIT	2	LCPR	A	MPES
3	SOOM	B	ZOMP	3	LCCC	B	DVES
4	SZOM	C	UIT	4	LCIA	C	RDIR
5	SCRB	D	LIMP	5	LCIE	D	WIR
6	EXMA	E	LOMP	6	LCIR	E	BRM
7	EXMA	F	SOMP	7	MBVV	F	BRMI

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	ABMM	ZBMM	OBMM	TBMM	ABMB	ZBMB	TBMB	CBMB	LDXT STXT LDMT STMT	NOP	IRRD TTRD	CRRT CRRQ TTRQ	ESD ESS	TRRQ LDXD	CRXD STXD	AUG 8F
9	ABSM	ZBSM	OBSM	TBSM	ABSB	ZBSB	TBSB	CBSB	ABXM	ZBXM	OBXM	TBXM	ABXB	ZBXB	TBXB	CBXB
A	MPM	DVM	MPRD MPMD	DVRD DVMD	LFM	SFM	HHI	AUG A7	HNS HNR	HZS HZR	HOS HOR	HCS HCR	HLS HGE	HLE HGT	LBX	SBX
B	MPS	DVS	SCCC	EXR	LFS	SFS	IRM	IRR	MPX	DVX	PLM	PSM	LFX	SFX	LDAM LDVM	STAM STVM
C	ADMM	ETMM	ORMM	CRM	ADMB	ETMB	TRMB	CRMB	ADRD ADMD	SURD SUMD	AUG CA	UIT	TSBM	TRRD LDMD	CLM STMD CLMD	CRRD CRMD
D	ADSM	ETSM	ORSM	CRS	ADSB	ETSB	TRSB	CRSB	ADXM	ETXM	ORXM	CRX	ADXB	ETXB	TRXB	CRXB
E	ADM	SUM	ETM	ORM	XOM	LDM	STM	BRU BLM	AUG E8	SUI CRI	ETI TETI	ORI TORI	XOI TXOI	LDI LDF LDFD LDFQ	STI	BLI
F	ADS	SUS	ETS	ORS	XOS	LDS	STS	HOP BLT	ADX	SUX	ETX	ORX	XOX	LDX	STX	BRX BLX

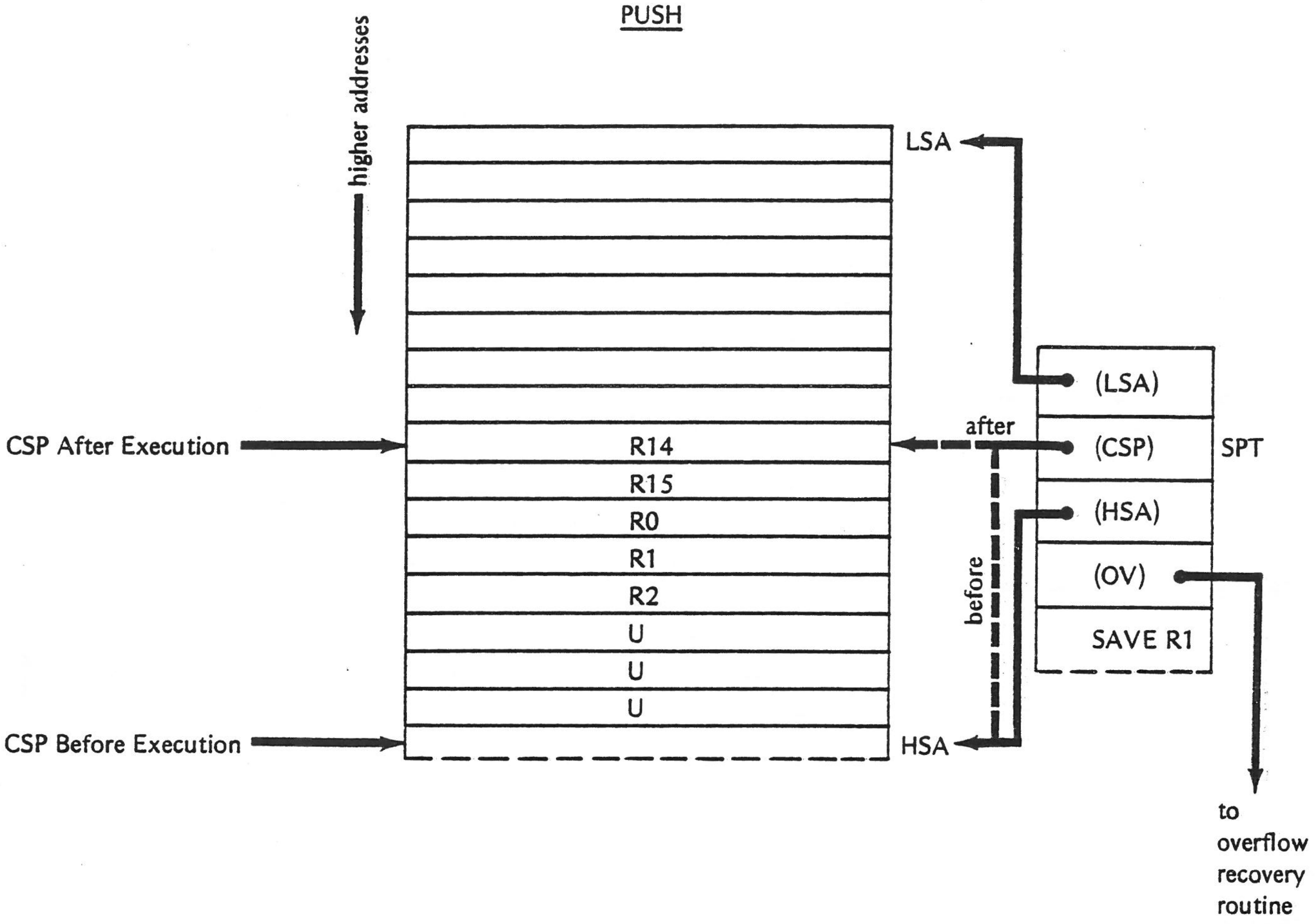
AUGMENTS

Ra	A7	Ra	A7	Ra	8F	Ra	8F	Ra	CA	Ra	CA	Rb	E8	Rb	E8
0	BLNS	8	BLNR	0	BXNS	8	BXNR	0	SRNS	8	SRNR	0	ADI	8	EPMD
1	BLZS	9	BLZR	1	BXZS	9	BXZR	1	SRZS	9	SRZR	1	LDES	9	CRZ
2	BLOS	A	BLOR	2	BXOS	A	BXOR	2	SROS	A	SROR	2	ADES	A	CRZD
3	BLCS	B	BLCR	3	BXCS	B	BXCR	3	SRCS	B	SRCR	3	SUES	B	XPMD
4	BLLS	C	BLGE	4	BXLS	C	BXGE	4	SRLS	C	SRGE	4	CIES	C	LTIL
5	BLLE	D	BLGT	5	BXLE	D	BXGT	5	SRLE	D	SRGT	5	EXI	D	LTDL
6	BLHI	E	BLNH	6	BXHI	E	BXNH	6	SRHI	E	SRNH	6	MPI	E	LTID
												7	DVI	F	LTDD

Appendix E. EXAMPLES OF FLOATING POINT NUMBERS

Number	*Machine Representation*	*Number*	*Machine Representation*
**** Single Precision*		**** Single Precision*	
16.000	41600000	1.000	40600000
15.000	413C0000	0.938	403C0000
14.000	41380000	0.875	40380000
13.000	41340000	0.813	40340000
12.000	41300000	0.750	40300000
11.000	412C0000	0.688	402C0000
10.000	41280000	0.625	40280000
9.000	41240000	0.563	40240000
8.000	41200000	0.500	40200000
7.000	40F80000	0.438	3FF80000
6.000	40F00000	0.375	3FF00000
5.000	40E80000	0.313	3FE80000
4.000	40E00000	0.250	3FE00000
3.000	40B00000	0.188	3FB00000
2.000	40A00000	0.125	3FA00000
1.000	40600000	0.063	3F600000
0.000	0	0.000	0
-1.000	BFA00000	-0.063	C0A00000
-2.000	BF600000	-0.125	C0600000
-3.000	BF500000	-0.188	C0500000
-4.000	BF200000	-0.250	C0200000
-5.000	BF180000	-0.313	C0180000
-6.000	BF100000	-0.375	C0100000
-7.000	BF080000	-0.488	C0080000
-8.000	BEE00000	-0.500	BFB00000
-9.000	BEDC0000	-0.563	BFDC0000
-10.000	BED80000	-0.625	BFD80000
-11.000	BED40000	-0.688	BFD40000
-12.000	BED00000	-0.750	BFD00000
-13.000	BECC0000	-0.813	BFCC0000
-14.000	BEC80000	-0.875	BFC80000
-15.000	BEC40000	-0.938	BFC40000

Appendix F. EXAMPLES OF PUSH/PULL INSTRUCTIONS

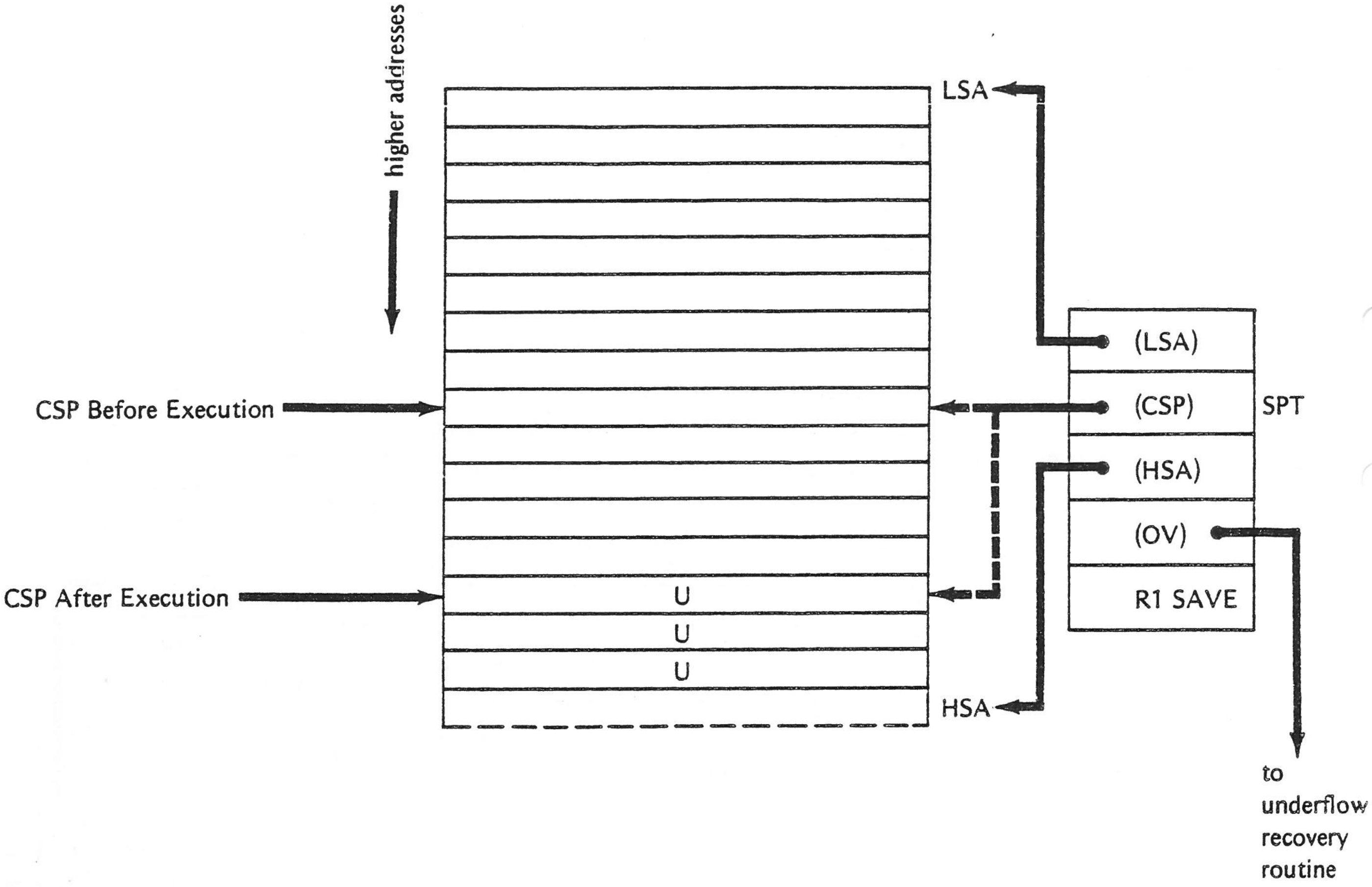


Given eight words are to be pushed and five registers are to be saved starting at R14, the above will occur.

NW = 8
NR = 5 (NR-1=4)
RA = 14

Before the PUSH instruction is executed, the stack is "empty" (that is CSP-HSA). After the instruction is executed, a space of eight words has been allocated in the stack (towards lower addresses). In the lowest five words of the allocated space the indicated registers are saved (towards higher addresses). The remaining three words of allocated space are unmodified.

PULL



Given the stack configuration of the PUSH example, five words are to be pulled and no registers are to be restored, the above will occur.

NW	=	5
NR	=	1 (NR-1=0)
RA	=	0

Register R0 ("bit bucket") is restored from the first word currently pointed to in the stack. A space of five words is then released by incrementing CSP. Three words remain allocated in the stack after the completion of the pull

Appendix G. HEXADECIMAL TO DECIMAL CONVERSION

This appendix enables direct conversion of decimal numbers to/from hexadecimal numbers in the ranges:

HEXADECIMAL	DECIMAL
000 to FFF	0000 to 4095

For numbers outside the range of the table, add the following values to the table figures:

HEXADECIMAL	DECIMAL	HEXADECIMAL	DECIMAL
1000	4096	9000	36864
2000	8192	A000	40960
3000	12288	B000	45056
4000	16384	C000	49152
5000	20480	D000	53248
6000	24576	E000	57344
7000	28672	F000	61440
8000	32768		

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
010	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
020	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
030	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
040	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
050	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
060	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
070	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
080	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
090	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A0	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B0	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C0	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D0	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E0	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F0	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
100	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
110	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
120	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
130	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
140	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
150	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
160	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
170	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
180	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
190	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A0	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B0	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C0	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D0	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E0	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F0	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511

200	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
210	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
220	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
230	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
240	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
250	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
260	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
270	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
280	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
290	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A0	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B0	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C0	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D0	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E0	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F0	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
300	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
310	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
320	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
330	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
340	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
350	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
360	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
370	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
380	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
390	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A0	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B0	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C0	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D0	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E0	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F0	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
400	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
410	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
420	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
430	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
440	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
450	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
460	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
470	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
480	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
490	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A0	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B0	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C0	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D0	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E0	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F0	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
500	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
510	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
520	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
530	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
540	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
550	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
560	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
570	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
580	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
590	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A0	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B0	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C0	1472	1473	1474	1475	1476	1477	1478	1779	1480	1481	1482	1483	1484	1485	1486	1487
5D0	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E0	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F0	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
600	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
610	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
620	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
630	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
640	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
650	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
660	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
670	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
680	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
690	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A0	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B0	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C0	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D0	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E0	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F0	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791
700	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
710	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
720	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
730	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
740	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
750	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
760	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
770	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
780	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
790	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A0	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B0	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C0	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D0	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E0	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F0	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
800	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
810	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
820	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
830	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
840	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
850	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
860	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
870	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
880	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
890	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A0	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B0	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C0	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D0	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E0	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F0	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
900	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
910	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
920	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
930	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
940	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
950	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
960	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
970	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
980	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
990	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A0	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B0	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C0	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D0	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E0	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F0	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A00	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A10	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A20	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A30	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A40	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A50	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A60	2656	2657	2658	2569	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A70	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A80	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A90	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA0	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB0	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC0	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD0	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE0	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF0	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B00	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B10	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B20	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B30	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B40	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B50	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B60	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B70	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B80	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B90	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA0	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB0	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC0	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD0	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE0	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF0	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C00	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C10	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C20	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C30	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C40	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C50	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C60	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C70	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C80	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C90	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA0	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB0	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC0	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD0	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE0	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF0	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
D00	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D10	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D20	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D30	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D40	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D50	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D60	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D70	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D80	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D90	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA0	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB0	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC0	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD0	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE0	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF0	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E00	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E10	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E20	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E30	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E40	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E50	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E60	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E70	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E80	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E90	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA0	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB0	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC0	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED0	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE0	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF0	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F00	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F10	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F20	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F30	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F40	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F50	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F60	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F70	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F80	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F90	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA0	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB0	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC0	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD0	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE0	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF0	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

Appendix H. TABLE OF POWERS OF TWO AND SIXTEEN

16^k			
2^n	n	k	2^{-n}
1	0	0	1.0
2	1		0.5
4	2		0.25
8	3		0.125
16	4	1	0.062 5
32	5		0.031 25
64	6		0.015 625
128	7		0.007 812 5
256	8	2	0.003 906 25
512	9		0.001 953 125
1 024	10		0.000 976 562 5
2 048	11		0.000 488 281 25
4 096	12	3	0.000 244 140 625
8 192	13		0.000 122 070 312 5
16 384	14		0.000 061 035 156 25
32 768	15		0.000 030 517 578 125
65 536	16	4	0.000 015 258 789 062 5
131 072	17		0.000 007 629 394 531 25
262 144	18		0.000 003 814 697 265 625
524 288	19		0.000 001 907 348 632 812 5
1 048 576	20	5	0.000 000 953 674 316 406 25
2 097 152	21		0.000 000 476 837 158 203 125
4 194 304	22		0.000 000 238 418 579 101 562 5
8 388 608	23		0.000 000 119 209 289 550 781 25
16 777 216	24	6	0.000 000 059 604 644 775 390 625
33 554 432	25		0.000 000 029 802 322 387 695 312 5
67 108 864	26		0.000 000 014 901 161 193 847 656 25
134 217 728	27		0.000 000 007 450 580 596 923 828 125
268 435 456	28	7	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29		0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30		0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31		0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	8	0.000 000 000 232 830 643 653 869 628 906 25

Appendix I.
ASCII INTERCHANGE CODE SET WITH ANSI CARD PUNCH CODES

Row	*Col.*	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
0000	0	NUL 12-0-9-8-1	DLE 12-11-9-8-1	SP No punch	0 0	@ 8-4	P 11-7	\ 8-1	p 12-11-7
0001	1	SOH 12-9-1	DC1 11-9-1	! 12-8-7	1 1	A 12-1	Q 11-8	a 12-0-1	q 12-11-8
0010	2	STX 12-9-2	DC2 11-9-2	" 8-7	2 2	B 12-2	R 11-9	b 12-0-2	r 12-11-9
0011	3	ETX 12-9-3	DC3 11-9-3	# 8-3	3 3	C 12-3	S 0-2	c 12-0-3	s 11-0-2
0100	4	EOT 9-7	DC4 9-8-4	$ 11-8-3	4 4	D 12-4	T 0-3	d 12-0-4	t 11-0-3
0101	5	ENQ 0-9-8-5	NAK 9-8-5	% 0-8-4	5 5	E 12-5	U 0-4	e 12-0-5	u 11-0-4
0110	6	ACK 0-9-8-6	SYN 9-2	& 12	6 6	F 12-6	V 0-5	f 12-0-6	v 11-0-5
0111	7	BEL 0-9-8-7	ETB 0-9-6	' 8-5	7 7	G 12-7	W 0-6	g 12-0-7	w 11-0-6
1000	8	BS 11-9-6	CAN 11-9-8	(12-8-5	8 8	H 12-8	X 0-7	h 12-0-8	x 11-0-7
1001	9	HT 12-9-5	EM 11-9-8-1	) 11-8-5	9 9	I 12-9	Y 0-8	i 12-0-9	y 11-0-8
1010	A	LF 0-9-5	SUB 9-8-7	* 11-8-4	: 8-2	J 11-1	Z 0-9	j 12-11-1	z 11-0-9
1011	B	VT 12-9-8-3	ESC 0-9-7	+ 12-8-6	; 11-8-6	K 11-2	[12-8-2	k 12-11-2	{ 12-0
1100	C	FF 12-9-8-4	FS 11-9-8-4	, 0-8-3	< 12-8-4	L 11-3	\ 0-8-2	l 12-11-3	¦ 12-11
1101	D	CR 12-9-8-5	GS 11-9-8-5	– 11	= 8-6	M 11-4	] 11-8-2	m 12-11-4	} 11-0
1110	E	SO 12-9-8-6	RS 11-9-8-6	. 12-8-3	> 0-8-6	N 11-5	∧ 11-8-7	n 12-11-5	~ 11-0-1
1111	F	SI 12-9-8-7	US 11-9-8-7	/ 0-1	? 0-8-7	O 11-6	_ 0-8-5	o 12-11-6	DEL 12-9-7

Some positions in the ASCII code chart may have a different graphic representation on various devices as shown below.

Teletype 33	*ASCII*	*IBM029*
!	!	¡
[	[	¢
]	]	!
↑	∧	¬
←	_	_

CONTROL CODING DEFINITIONS

Control Characters:

NUL	– Null
SOH	– Start of Heading (CC)
STX	– Start of Text (CC)
ETX	– End of Text (CC)
EOT	– End of Transmission (CC)
ENQ	– Enquiry (CC)
ACK	– Acknowledge (CC)
BEL	– Bell (audible or attention signal)
BS	– Backspace (FE)
HT	– Horizontal Tabulation (punch card skip) (FE)
LF	– Line Feed (FE)
VT	– Vertical Tabulation (FE)
FF	– Form Feed (FE)
CR	– Carriage Return (FE)
SO	– Shift Out
SI	– Shift In
DLE	– Data Link Escape (CC)
DC1	– Device Control 1
DC2	– Device Control 2

Control Characters:

DC3	– Device Control 3
DC4	– Device Control 4 (stop)
NAK	– Negative Acknowledge (CC)
SYN	– Synchronous Idle (CC)
ETB	– End of Transmission Block (CC)
CAN	– Cancel
EM	– End of Medium
SS	– Start of Special Sequence
ESC	– Escape
FS	– File Separator (IS)
GS	– Group Separator (IS)
RS	– Record Separator (IS)
US	– Unit Separator (IS)
DEL	– Delete
SP	– Space (normally nonprinting)
(CC)	– Communication Control
(FE)	– Format Effector
(IS)	– Information Separator

Appendix J. SYSTEM DIFFERENCES

This appendix describes the various operating characteristics of the Classic 7860 Series Computer which are not compatible with the MODCOMP IV Computer.

MODCOMP IV Instructions Not Supported by the Classic

AMEM	Allocate Memory Pages to Map Image
DMEM	Deallocate Memory Pages from Map Image
MABI	Memory Allocation Block Initialize
SGP	Set Global Protect Register
SLP	Set Lower Protect Register
SPR	Set Protect Register
SUP	Set Upper Protect Register

Classic Instructions Not Supported by the MODCOMP IV

ADES	Add Immediate with Extended Sign
BRM	Branch to Microroutine
BRMI	Branch to Microroutine Immediate
CDFI	Convert Double Precision Floating Point Operand to Double Integer
CIES	Compare Immediate with Extended Sign
CLM	Clear Memory
CLMD	Clear Memory Doubleword
CQFI	Convert Quad Precision Floating Point Operand to Double Integer
CRRQ	Compare Quad Register to Quad Register
CRRT	Compare Triple Register to Triple Register
CRXD	Compare Double Register to (Short Indexed) Memory Doubleword
CRZ	Compare Register to Zero
CRZD	Compare Double Register to Zero
DVES	Divide Immediate with Extended Sign
DVI	Divide Immediate
EPMD	Enter Pipeline Mode of Execution
EXI	Execute Immediate
EXMA	Enter Extended Memory Addressing Mode
FAI	Floating Point Add Immediate
FAID	Floating Point Add Immediate Double Precision
FAIQ	Floating Point Add Immediate Quad Precision
FDI	Floating Point Add Divide Immediate Quad Precision
FDID	Floating Point Divide Quad Precision
FDIQ	Floating Point Divide Quad Precision
FMI	Floating Point Multiply Immediate Quad Precision
FMID	Floating Point Multiply Immediate Quad Precision
FMIQ	Floating Point Multiply Immediate Quad Precision
FSI	Floating Point Subtract Immediate Quad Precision
FSID	Floating Point Subtract Immediate Double Precision
FSIQ	Floating Point Subtract Immediate Quad Precision
IRRD	Interchange Double Register and Double Register
ISZ	Input Status from Device Zero
LDES	Load Immediate and Extend Sign
LDF	Load Immediate Floating Point
LDFD	Load Immediate Floating Point Double Precision
LDFQ	Load Immediate Floating Point Quad Precision
LDMI	Load Triple Register from Memory Tripleword
LDXT	Load Triple Register from Memory Tripleword (Short Indexed)
LTDD	Loop Termination with Directly Addressed Control Variable and Directly Addressed Terminal Value
LTDL	Loop Termination with Directly Addressed Control Variable and Literal Terminal Value
LTID	Loop Termination with Indirectly Addressed Control Variable and Directly Addressed Terminal Value
LTIL	Loop Termination with Indirectly Addressed Control
LXR	Load Extended Memory Control Register
MBEV	Move Block from Extended to Virtual Memory
MBVE	Move Block from Virtual to Extended Memory
MBVV	Move Virtual Block to Virtual Block
MPES	Multiply Immediate with Extended Sign
MPI	Multiply Immediate
RDI	Read Internal Registers
RMFS	Read Memory Plane Status
RMWS	Read Memory Word Status
SRCR	Set Register it Condition Code C Reset
SRCS	Set Register it Condition Code C Set
SRGE	Set Register on Greater than or Equal Condition
SRGT	Set Register on Greater than Condition
SRHI	Set Register on Magnitude Higher Condition
SRLE	Set Register on Less than or Equal Condition
SLRS	Set Register on Less than Condition
SRNH	Set Register on Magnitude Not Higher Condition

SRNR	Set Register Condition Code N Reset
SRNS	Set Register on Condition Code N Set
SROR	Set Register on Condition Code O Reset
SROS	Set Register on Condition Code O Set
SRZR	Set Register on Condition Code Z Reset
SRZS	Set Register on Condition Code Z Set
STMT	Store Triple Register into Memory Tripleword
STXT	Store Triple Register into Memory Tripleword (Short Indexed)
SUES	Subtract Immediate with Extended Sign
TSBM	Test and Set Bit in Memory
WRI	Write Internal Register
WMS	Write Memory Status
XPMD	Exit Pipeline Mode of Execution
XVMO	Exit Virtual Mode of CPU Execution

Additional differences in operating characteristics:

- During traps the Classic condition codes are strobed with additional information regarding the trap condition. This information is documented in Chapter 3 of this manual.
- During execution of all Classic shift instructions, a shift count of zero will cause condition code C to be reset, The MODCOMP IV maintains condition code C unaffected for zero shift counts.
- The manner in which the Classic calculates "greater than or equal to" during conditional operation instructions (example, HGE) differs slightly from the MODCOMP IV:
  Classic: GE = CCZ V (CCZ and (CCN XOR CCO))
  MODCOMP IV: GE = CCN XOR CCO -1
- The Classic features a global parity error detection capability by enabling the processor's interrupt level 1. Level 1 is permanently enabled in the MODCOMP IV. Level 1 disabled in the Classic will inhibit global error detection but will not affect CPU MPE detection, which generates a level 1 trap. See Chapter 3 for a detailed description of global error reporting.
- The MODCOMP II/III protection scheme is unimplemented in the Classic. SGP, SLP, SPR and SUP will no-op if attempted.
- The Classic CPU features a more deeply pipelined memory interface than the MODCOMP IV, which accelerates fetching of consecutive instruction words. Accordingly, software should not attempt to modify its instruction space between PR and PR+8, as these locations may have already been fetched by the CPU.
- The results of floating point calculations in the Classic are rounded to the desired precision instead of truncated as in the MODCOMP IV.
- Because of the different integer divide algorithm used by the Classic, an integer divide result which overflows and would therefore be erroneous in either case, will be numerically different in the Classic than in the MODCOMP IV.
- The Classic DMPI instruction requires 6 bits (64 DMP Channels) to specify a channel number as opposed to the MODCOMP IV which requires 5 bits, (32 channels) and the IV/25 which requires 4 bits (16 channels). To accomodate the 6 bit channel number, the Classic DMPI instruction is formatted with channel number in register R of the register pair specified in the instruction. See Chapter 8 for a description of DMPI.
- Only Class II and Class III controllers are supported by the Classic.
- The Classic automatically executes DMP I/O in the relocatable mode when the CPU is in Virtual Mode and in non-relocatable mode when the CPU is in non-virtual mode.
- As described in Chapter 3, each memory module possesses six unique status registers not present in the MODCOMP IV memory system. These registers latch error information upon occurence of a parity error.
- The Classic memory power supply provides an early warning power failure indication that gives the user an option of responding to an impending power outage at an earlier point in time. In the worst case, the early warning power fail interrupt will occur 6 ms before the final power fail interrupt.

INDEX

User Comments

We at Modular Computer Systems feel that the comments and suggestions of our readers provide the best indication of the quality and effectiveness of our publications. Therefore, your input is essential in our effort to publish the most complete and accurate information possible. Please take a few seconds to fill out the post-paid card and help us to provide the documentation that you need.

For additional documentation contact the Modular Computer Systems Sales Office in your area

Fold

FIRST CLASS
PERMIT NO. 3624
FORT LAUDERDALE,
FLORIDA

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY...

MODULAR COMPUTER SYSTEMS
P.O. BOX 6099/1650 W. McNAB ROAD
FT. LAUDERDALE, FLORIDA 33309

Attention: Publications, Mail Stop 30

Fold

MODCOMP

Publication No. ______________________

Did you find this publication complete, accurate, readable?

Comment: ______________________

Did you find errors in this publication? (Please specify page No.)

Do you have any general suggestions to improve this publication?

Name ______________________ Position ______________________

Company ______________________

Address ______________________